**COMPONENT BASED SIMULATION OF THE SPACE OPERATIONS
VEHICLE AND THE COMMON AERO VEHICLE**
Phillip E. Pournelle Lieutenant, United States Navy
B.A., San Diego State University, 1991

MASTER OF OPERATIONS ANALYSIS-MARCH 1999

Thesis Advisor: Arnold H. Buss, Department of Operations Analysis
Second Reader: Charles H. Shaw, III, Operations Analysis

In the ongoing competition between strike platforms and Integrated Air Defense Systems (IADS), strike platforms currently have the upper hand. However, the desire of modern military and political leaders to conduct accurate and effective strike missions on enemy military targets using limited resources, with a minimal risk to U.S. resources and personnel, in a timely manner is in direct competition with an adversary's goal of defending his territory and assets. An adversary can threaten, deter or even destroy strike assets used to attack him using IADS, aircraft and other means. With fewer overseas bases and increasing demands on the Carrier Battle Groups (CVBGs), it may be necessary to look at new technologies to enable the United States to strike potential opponents in a timely fashion. The Space Operations Vehicle (SOV) is a low earth orbit capable space vehicle being developed by Phillips Laboratories at Kirtland AFB, New Mexico. The SOV will be a cross between the space shuttle and an F-14 fighter, a rugged low earth orbit capable vehicle designed to conduct multiple sorties for military purposes.

This thesis develops a software component architecture and component library for building simulations for analysis of current and proposed military systems such as the SOV. With this software package, proposed and current weapon systems performances can be simulated, tested, evaluated and adjusted in an iterative process by which analysts can make modifications to the simulation with greater speed, flexibility and productivity. This software package will support the Simulation Based Acquisition (SBA) process through all the phases and milestones of an appropriation program and help ensure the United States obtains the best equipment to maintain its edge on the battlefield.

This package is ideal for the initial evaluation of concepts and plans and is poised for growth as a comprehensive analysis tool for the entire acquisition process.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**COMPONENT BASED SIMULATION OF
THE SPACE OPERATIONS VEHICLE AND
THE COMMON AERO VEHICLE**

by

Phillip E. Pournelle

March 1999

Thesis Advisor: Arnold H. Buss
Second Reader: Charles H. Shaw, III

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE **Component Based Simulation of The Space Operations Vehicle and The Common Aero Vehicle**			**5. FUNDING NUMBERS**
6. AUTHOR(S) Pournelle, Phillip E.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) **Air Force Research Laboratory** **Kirtland Air Force Base, New Mexico**			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

The desire of modern military and political leaders to conduct accurate and effective strike missions on enemy military targets using limited resources, with a minimal risk to U.S. resources and personnel, in a timely manner is in direct competition with an adversary's goal of defending his territory and assets. An adversary can threaten, deter or even destroy strike assets used to attack him using IADS, aircraft and other means New technologies may enable the United States to strike potential opponents in a timely fashion. The Space Operations Vehicle (SOV) is a low earth orbit capable space vehicle being developed by Phillips Laboratories at Kirtland AFB, New Mexico. The SOV will be a cross between the space shuttle and an F-14 fighter, a rugged low earth orbit capable vehicle designed to conduct multiple sorties for military purposes.

This thesis develops a software component architecture and component library for building simulations for analysis of current and proposed military systems such as the SOV. This software package will support the Simulation Based Acquisition (SBA) process through all the phases and milestones of a program and help ensure the United States obtains the best equipment to maintain its edge on the battlefield.

14. SUBJECT TERMS Component Based Simulations, Simulation Based Acquisition, Space Operations Vehicle, Common Aero Vehicle, Air Defense Systems, Strike Warfare.			**15. NUMBER OF PAGES** 131
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFI- CATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

COMPONENT BASED SIMULATION OF THE SPACE OPERATIONS VEHICLE AND THE COMMON AERO VEHICLE

Phillip E. Pournelle - Lieutenant, United States Navy
B.A., San Diego State University, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE OPERATIONS RESEARCH

from the

**NAVAL POSTGRADUATE SCHOOL
March 1999**

Author: _____

Phillip E. Pournelle

Approved by: _____

Arnold H. Buss, Thesis Advisor

_____

Charles H. Shaw, III, Second Reader

_____

Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

In the ongoing competition between strike platforms and Integrated Air Defense Systems (IADS), strike platforms currently have the upper hand. However, the desire of modern military and political leaders to conduct accurate and effective strike missions on enemy military targets using limited resources, with a minimal risk to U.S. resources and personnel, in a timely manner is in direct competition with an adversary's goal of defending his territory and assets. An adversary can threaten, deter or even destroy strike assets used to attack him using IADS, aircraft and other means. With fewer overseas bases and increasing demands on the Carrier Battle Groups (CVBGs), it may be necessary to look at new technologies to enable the United States to strike potential opponents in a timely fashion. The Space Operations Vehicle (SOV) is a low earth orbit capable space vehicle being developed by Phillips Laboratories at Kirtland AFB, New Mexico. The SOV would be a cross between the space shuttle and an F-14 fighter, a rugged low earth orbit capable vehicle designed to conduct multiple sorties for military purposes.

This thesis develops a software component architecture and component library for building simulations for analysis of current and proposed military systems such as the SOV. With this software package, proposed and current weapon systems performances can be simulated, tested, evaluated and adjusted in an iterative process by which analysts can make modifications to the simulation with greater speed, flexibility and productivity. This software package will support the Simulation Based Acquisition (SBA) process through all the phases and milestones of a program and help ensure the United States obtains the best equipment to maintain its edge on the battlefield. This package is ideal for the initial evaluation of concepts and plans and is poised for growth through the entire acquisition process.

THESIS DISCLAIMER

The reader is cautioned that the computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

In the ongoing competition between strike platforms and Integrated Air Defense Systems (IADS), strike platforms currently have the upper hand. However, the desire of modern military and political leaders to conduct accurate and effective strike missions on enemy military targets using limited resources, with minimal risk to U.S. resources and personnel, in a timely manner is in direct competition with an adversaries goal of defending his territory and assets. An adversary can threaten, deter or even destroy strike assets used to attack him using IADS, aircraft and other means. With fewer overseas bases and increasing demands on the Carrier Battle Groups (CVBGs), it may be necessary to look at new technologies to enable the United States to strike potential opponents in a timely fashion. The Space Operations Vehicle (SOV) is a low earth orbit capable space vehicle being developed by Phillips Laboratories at Kirtland AFB, New Mexico. The SOV would be a cross between the space shuttle and an F-14 fighter, a rugged low earth orbit capable vehicle designed to conduct multiple sorties for military purposes. Other missions for the SOV include satellite replacement, high altitude reconnaissance, space asset protection, space asset attrition and low earth orbit dominance.

High technology concepts such as the SOV require a high level of evaluation during the entire course of its lifetime. Costly systems must be carefully evaluated during the Concept Exploration phase and built upon during the remaining acquisition process. The Simulation, Test and Evaluation Process (STEP) is now an integral part of the Test and Evaluation Master Plan (TEMP). This requirement from the Under Secretary of Defense for Acquisition and Technology demands that all future acquisitions of high technology systems use an iterative process where simulations are conducted, followed by tests and assessments. The results of each iteration is to be used in the next round of simulations and tests.

The Simulation Based Acquisition (SBA) requirement is intended to ensure the United States Military obtains the most effective technology at the lowest cost. However SBA requires flexible, adaptable and alterable simulations that are currently not available.

This thesis develops a software component architecture and component library for building simulations for analysis of current and proposed military systems. With this software package, proposed and current weapon systems performances can be simulated, tested, evaluated and adjusted in an iterative process by which analysts can make modifications to the simulation with greater speed, flexibility and productivity. This software package will support the SBA process through all the milestones and phases and help ensure the United States obtains the best equipment to maintain its edge on the battlefield. This package is ideal for the initial evaluation of concepts and plans and is poised for growth through the entire acquisition process.

ACKNOWLEDGEMENTS

I.   INTRODUCTION

In the opening salvoes of Operation Desert Storm, allied forces devoted the first of many missions to the Suppression of Enemy (Iraqi) Air Defenses (SEAD). Completion of these missions gave allied forces freedom of action to pursue military targets and shape the battlefield for the future successful ground campaign. Success of the SEAD and strike missions was greatly enhanced by the months of preparation to bring the allied air armada together in nearby states.

Since the Gulf War, the National Command Authority (NCA) has directed the use of military power through new Force Projection systems to strike targets in other countries during Operations Other Than War (OOTW) without significant time to build up forces or plans. Weapons like cruise missiles give us the ability to penetrate enemy air defenses without having to fight through them or risk pilots lives. However, cruise missiles are expensive, few in number and have limited ability to strike targets that move or are heavily fortified.

The desire of modern military and political leaders to project force through accurate and effective strike missions on enemy military targets, using limited resources, with a minimum of risk to United States (U.S.) resources and personnel is in direct competition with an adversary's goal of defending his territory. Armed with an Integrated Air Defense System (IADS) including aircraft, Surface to Air Missiles (SAMs) and Anti-Aircraft Artillery (AAA); the defender can deter or even destroy assets an attacker brings to bear.

While the Air Force's Air Expeditionary Force (AEF) can bring to bear weighty ordnance from heavy bombers, it has a long cycle time between missions, particularly if it must travel from the Continental United States (CONUS). The AEF is further limited since the U.S. has fewer forward deployed forces, overseas air bases, and more restrictions on missions and over-flights. With these restrictions our leaders turn to the United States Navy's Carrier Battle Groups (CVBG) to take up those missions. However, the CVBG takes time to reach the operational area where it can effectively launch its forces. As the number of CVBGs continues to decline, the time they take to begin their

missions increases and the depletion of the manpower and equipment increases due to the high Operations Tempo (OPTEMPO).

A potential third strike alternative is being developed that would be able to bypass the enemy's air defenses, penetrate heavily defended and fortified targets, hit a target anywhere on the planet within thirty minutes of launch, never expose personnel to enemy fire and not be restricted by basing or over-flight rights.   That alternative is the SOV, a low earth orbit capable space vehicle being developed by Phillips Laboratories at Kirtland AFB, New Mexico.   The SOV would be a cross between the space shuttle and an F-14 fighter. It is a rugged, low earth orbit (LEO) capable vehicle designed to conduct multiple sorties for military purposes.

The major problem associated with this effort is we don't have an SOV to experiment with. It is still in the conceptual design phase. While we can generate mission needs and operational requirements for such a system allowing us to identify basic constructs, we need to be able to integrate changes to the model and incorporate any test results or insight quickly and easily. Further, as we gain data from tests and experiments, we need to be able to incorporate them into our analysis program to gain further insight. We need a comprehensive program and enhancement process to analyze new weapons concepts. This requirement has lead to the concept of Simulation Based Acquisition (SBA).

A.   SIMULATION BASED ACQUISITION

The United States military must take additional measures to ensure it acquires the best weapons systems to accomplish their missions in a shrinking budget environment combined with ever increasing mission needs.

One mechanism employed for this purpose is Simulation Based Acquisition (SBA). Using SBA, Services can compare existing or proposed weapons systems while reducing the cost of obtaining prototypes or test models and conducting "fly-offs" between them. Simulations can also be varied in differing environments to assist evaluations over a range of possible working environs.   However, re-useable SBA simulations do not exist and are generally created on the fly as a single use test project (Coyle, 1997)

On October 3, 1995 the Under Secretary of Defense for Acquisition and Technology announced that all future acquisition programs for military equipment shall integrate simulation into every step in the process. "This means our underlying approach will be to model first, simulate, then test, and then iterate the test results back into the model." (STEP Guidelines, 1997).

To accomplish this, analysts will require simulations they can quickly modify, test, evaluate and re-use with further changes. The Simulation, Test and Evaluation Process (STEP) requires flexible model kits that allow analysts to be involved in the acquisition process from Concept Development and Initial Design through Deployment and follow-on improvements.

A Component Software Design for these models and simulations can meet these challenges. By using small, loosely coupled components we can create a simulation where we can measure the performance of an object's behavior, their interaction with each other and overall mission effectiveness without creating large models that become unwieldy and difficult to modify.

To simulate and evaluate the SOV/CAV and to ensure future exploration, this thesis will expand the work done in Component Software Design and show how it can be used to compare existing and future strike warfare designs.

B.   HISTORICAL BACKGROUND

This thesis argues that the SOV is an inevitable development of space technology and should be rationally examined as an alternative to other conventional strike platforms. This section will present a discussion of the constant battle between air power and air defense systems, the idea of two stage combat systems, and the technological advances that affect them.

1.   The competition between Air Forces and ground based Air Defenses

In 1914 the Red Baron could have been shot down by ground fire (Angelucci, 1973). A lowly infantryman firing a rifle or machine gun into the air as the Baron dueled with an allied aircraft could have brought down the vaunted air ace. This rifle shot was

the opening salvo in the long-running competition between surface anti-aircraft weapons and aircraft that continues to this day.

In World War II, the Imperial Japanese Navy's airforce gained the upper hand using dive bombers against allied ships; but, this was quickly reversed by 1942 beginning with the Battle of the Coral Sea where U.S. Navy ships firing shells with proximity fuses devastated the Japanese air forces (Hughes, 1986). Meanwhile, the Battle of Britain was decided in large part by the successful application of radar and associated Tactics, Techniques and Procedures (TTPs) as well as the early use of Operations Analysis (OA) by Sir John Blackett's "Flying Circus" to maximize defensive counter air (Terraine, 1988) (Arntzen, 1998).

In World War II, the Imperial Japanese Navy's air forces stunned the world by demonstrating that aircraft flying off a carrier could destroy enemy defenses while leaving the vulnerable portions of its forces safely behind. The application of the new flight technology to existing ship platforms demonstrated what Captain Wayne Hughes terms a two stage system (Hughes, 1993, 1995). By placing the infrastructure, ordnance and maintenance crew on the first stage such as an aircraft carrier, the airplane becomes a sleeker, smaller and more compact attack platform as a second stage while retaining is supporting infrastructure nearby. By fighting on the line in front of the first stage, it prevents its support structure from being attacked.

The development of carrier aircraft and appropriate TTPs to employ them created a revolutionary change from previously accepted methods of fighting. While the development of proximity fusing had a dramatic effect for a short period of time, it was an evolutionary development from the cannons pressed into service in earlier wars.

In the modern era, attacking aircraft have gained the upper hand in Suppression of Enemy Air Defenses (SEAD) through the use of Precision Guided Munitions (PGMs) and Anti-Radiation Missiles (ARMs) as well as Low Observable technology (LO or stealth). As Major Arent Arntzen argues in his thesis, this status may be only passing due to the development of passive sensors and other systems (Arntzen, 1998). This demonstrates that technology in itself favors neither the offense nor the defense in the long run (Van Creveld, 1989), but favors those willing to exploit it (Possony, 1978).

2.   Reusable Launch Vehicles

The application of radar, accompanying TTPs and air power in controlling the skys over Britain was revolutionary. The time may be ripe to do the same thing with another developing technology, Single Stage to Orbit (SSTO) rocket technology. SSTO appears to be a near term technology. An SSTO system is a single, reusable rocket system capable of delivering a payload into orbit or sub-orbit. In 1996 the X-31 Graham - Clipper ship was launched as a demonstration vehicle to prove that an SSTO system was controllable and reusable. It flew 9 sub-orbital flights and demonstrated that a rocket can be flown, refurbished, and re-launched in a 48 hour period. The X-33 Venture Star contract is a public/private joint venture to develop the system further. Companies such as Rotary Rocket are exploring their own approaches to this technology.

At the current rate, we may be seeing routine private space launches within the next five years. Just as the airplane became a dominant military platform after it matured as a civilian technology, the SSTO rocket may very soon be available for military missions.

C.   INITIAL COMPARISON OF STRIKE PLATFORMS

Before analysis can begin, a number of terms must be defined. Since the goal of this thesis is to compare the SOV to a CVBG and an AEF, we need to note their composition , their operational characteristics and their potential limitations.

1.   Platforms and Doctrine

a)   Carrier Aviation

The standard Carrier Battle Group (CVBG) consists of a Nimitz (CVN-68) class aircraft carrier, two Ticonderoga (CG-47) class cruisers, two Arleigh Burke (DD-51) class destroyers, and two Spruance (DD-963) class destroyers supported by a fast combat support ship (AOE) . All of the cruisers and destroyers carry varying numbers of Tomahawk missiles (for a total of 212) while the aircraft carrier bears the Carrier Air Wing (CVW) (Williams, 1995).

The CVW consists of 14 F-14D Tomcat air superiority fighters, 36 F/A-18 E/F Super Hornet multi-mission attack fighters, 4 EA-6B Prowler electronic

warfare aircraft, 8 S-3 Viking anti-submarine/electronic support aircraft, and 4 E-2C Hawkeye early warning aircraft. The carrier, as well as the other ships, also has a number of multi-mission helicopters to prosecute submarines, support logistics, conduct Combat Search and Rescue (CSAR) missions, etc.

The F-14D Tomcat (also known as the "Bombcat") and the F/A-18 Super Hornet are the two aircraft that primarily perform strike missions for the CVGB. The EA-6B Prowler is designed primarily for SEAD and can fire Anti-Radiation Missiles at active radars. The S-3 Viking is capable of supporting missions by monitoring enemy active emissions and can provide re-fueling support for any strike mission. The E-2C Hawkeye has an air search radar designed to provide airborne surveillance for the CVBG or any strike group to which it is assigned. While there are future designed aircraft on the drawing board, these units are projected to be a part of the CVBG for at least the next ten years.   As a composite squadron, the CVW is trained to work together and with the CVBG prior to deployment to deliver a multi-mission capable unit to Commander's In Chief (CINCs) around the world. The CINCs can then call upon the CVBG to respond to threats, or provide forward presence or be a flexible deterrent force in support of NCA objectives.

The CVBG can steam at a steady speed greater than 20 knots to within striking distance of most parts of the world. Plans are to maintain 11 CVBGs available for rotation to CINCs over the next ten years.

b)   Air Force Expeditionary Air Wing

After seeing the success of the CVBG and the CVW composite design, the Air Force developed the AEF to meet its rotating exigency around the world. Designed to be able to deploy to any friendly air field around the world within 48 hours, the AEF is a self contained unit consisting of combat aircraft, support aircraft, and aerial refueling. Additional forces such as bombers or other aircraft can be added as needed to meet operational requirements.

The core of the AEF is centered around 8 F-15C Eagle Air Superiority fighters, 8 F-16C Falcon night attack fighters, 8 F-15 Strike Eagle ground attack

fighters and 8 F-16CC Falcon SEAD fighters. For the purposes of a strike mission, 8 B-1B Lancer bombers can be attached, while 8 B-52H Stratofortresses can be attached to provide general bombardment or cruise missile support. Bombers can fly directly from CONUS and attack NCA designated targets; however, they are usually not deployed against an enemy with air defenses and Combat Air Patrols (CAP) active. In those cases, a full AEF may be needed to escort them.

While there are other forces the AEF may call on, such as stealth aircraft, Air Force policy prevents the analysis of such aircraft at this level of classification and they often may not be available for an AEF deployment.

c) Space Operations Vehicle armed with Common Aero Vehicle

The SOV is a low earth orbit (LEO) capable space vehicle being developed by Phillips Laboratories at Kirtland AFB, New Mexico (Military Spaceplane Program Office, 1997).   The SOV is a cross between the space shuttle and an F-14 fighter, a rugged LEO capable vehicle designed to conduct multiple sorties for military purposes. The SOV is designed to be a reusable space vehicle enabling military planners to conduct sustained space operations such as replacing satellite assets, space based reconnaissance, etc.   The SOV will be launched from a CONUS Air Force facility, conduct a mission, return to an airfield, and be reloaded and prepared for another launch within a few hours.

Current concepts call for the SOV to be launched from an airfield such as Minot AFB, reach apogee, release a booster called the Modular Insertion Stage (MIS), and return to earth at a downrange facility such as Eglin AFB or Holloman AFB. During a strike mission, the SOV would never enter orbit but deploy its package and return to earth. At the landing site, the SOV would be re-armed and refueled for another mission. This cycle of launching from one facility and landing at another would enable the SOV to conduct multiple missions in a sustained manner.

The MIS is a programmable booster designed to boost payloads from the SOV into orbit. If a mini-satellite was needed for surveillance or to temporarily replace a Global Positioning Satellite (GPS) unit, the MIS would be used to place the min-satellite into its proper orbit. For a strike mission, the MIS would be released at apogee to boost the Common Aero Vehicle (CAV) down range to re-enter the earth's atmosphere and conduct its strike mission.

The CAV is a delivery vehicle designed to be launched on the SOV, a conventional ballistic missile, an expendable launch vehicle such as a Delta rocket, or next generation expendable launch vehicle. The CAV is a reentry shell designed to separate from the launch vehicle, using a booster such as the MIS, begin a reentry maneuver, conduct maneuvers to bleed off kinetic energy and then deploy internally stored ordnance to conduct an attack. The standard CAV is designed to carry conventional self-guided ordnance available to any aircraft. A variant of the CAV is a unitary penetrator designed to conduct a high velocity penetration on hardened targets. This is expected to be able to penetrate a minimum of 30 to 40 feet of reinforced concrete.

For the purposes of this thesis, a squadron of twelve SOVs will be available for missions conforming to the Military Spaceplane Program Office's Systems Requirement Document for a Space Operations Vehicle System (Air Force Research Laboratory, 2 April 1998).

2.   Suppression of Enemy Air Defenses

To increase the probability of success of a mission and minimize the chances of the loss of aircraft and personnel, a number of missions must be directed towards the destruction or SEAD. Whether a portion of missions are directed to destroy the enemy forces in preliminary missions or suppression assets are devoted during the execution of a mission, the more resources devoted to that end will reduce the number of aircraft expected to be damaged or destroyed. While analytical models can be developed for some simple cases (see Appendix A), the operational environment is generally too complicated for these models to provide sufficient insight into the effectiveness of strike missions.

We can further reduce the exposure of aircraft and personnel by employing Over The Horizon (OTH) weapons. While the chances of an air defense battery detecting and engaging an OTH weapon is smaller than that of detecting and engaging an aircraft, weapons dropped from altitude and directed by a pilot have a greater chance of hitting a target than a cruise missile fired at where a target is thought to be.

The best of both worlds would be to incorporate stealth systems that allow the pilot to be on station directing accurate weapons while reducing his vulnerability to a small probability. The difficulty is that in the future, newer sensors will start to drive these numbers up. Today we have the restriction of operating all our stealth aircraft at night thus reducing the number of possible missions that can be done.

The SOV may become the ultimate in the two stage concept. By maintaining its base facilities within CONUS, its logistics base is very safe while the CAV can reach almost any point around the globe within thirty minutes. Additionally, by entering enemy air space directly above the target, it bypasses most of the stages of air defenses described earlier. High altitude air defense systems may be able to defend against it; but, the CAVs small size and high speed give it many of the advantages of stealth weapons. By bypassing the outer ring of defenses, the SOV/CAV combination can devote more sorties directly against the intended target.

3.   Time to Station Versus Sortie Cycle Time

In addition to the issue of how many missions to devote to SEAD and CAP we have to consider where the first stage of our operations exists. Carrier aircraft fly from the CVBG while Air Force aircraft must fly from a base facility.

a)   CVBG

The CVBGs circulate continuously from their homeports to areas of concern around the world. It will take time for them to arrive on station since they cannot stay on station indefinitely. If there is intelligence that something is brewing, they can be brought into a trouble spot as a sign of American resolve and be ready when the action begins. This means that it is not just our NCA who asks "where are the carriers?"

b)   AEW

An AEW can be deployed rapidly; but, it requires time to set up at its new base. They can also conduct a strike mission directly from CONUS; but, this creates a much longer cycle time. Either way, it is completely dependent on air base facilities.

In some cases, the base facility may be in the theater of operations (Such as Saudi Arabia during Desert Storm). This creates issues of protecting those air fields and the host allowing certain missions to occur. If the host nation is the one being attacked, they will most likely grant Operating Rights.   On the other hand, those facilities may be overrun. Other air fields may be in an adjacent area (such as air fields on Diego Garcia during Desert Fox); but, they add to the distance the strike force must fly for each mission and multiple missions require transport aircraft or ships to bring in more munitions.

c)   Space Operations Vehicle / Common Aero Vehicle

The SOV operates directly from its home base in CONUS. It requires no set up time to either arrive on station or steam into position. It requires no host country permission, only NCA tasking. The cycle time between flights of the

SOV/CAV is projected to be from 8 to 18 hours between flights for sustained wartime operations.

4. Point Targets versus Multiple Targets

The issues discussed above come into sharp focus if the NCA had an urgent need to strike a high value strategic target. Facilities such as suspected Weapons of Mass Destruction (WMD) production centers may be both time sensitive (because the material in question can be moved) and heavily defended. Such a point target would be a prime candidate for a stealth weapon system or OTH weapons system. All these factors extend the preparation time for such a mission. If follow-on attacks are necessary to ensure effectiveness, then the cycle time becomes an issue as well.

If the target nation sees a CVBG steaming towards its shore it may move the material. On the other hand, watching an AEW being deployed overseas on CNN may alert the enemy as well. The launch of a SOV/CAV would definitely get someone's attention, but they would have only thirty minutes to respond. As all three assets are flown often and regularly, a potential target nation may not be able to figure out which sortie is destined for them.

The preference order would appear to be CAV/SOV, AEW, and then CVBG against a point target if the goal is minimizing the time between the authorization order and hitting the target. The last two platforms are completely dependent on airfield location.

In the case of targets with multiple aim points, such as an airfield, the probability all aircraft equipment is moved before preparations are completed is not as critical. The cycle time of such a mission becomes critical as the number of aim points becomes large, as is the need for multiple follow up attacks to ensure confidence that the aircraft facilities are destroyed.

Over time, the SOV/CAV's lower sortie cycle rate may very well lose out against the CVBG or the AEW. However, it would have to be a significant campaign, since the SOV/CAV does not have to strip away the enemy's outer air defenses before continuing the campaign.

D.   PROBLEM STATEMENT

With the challenges of enemy air defenses in mind and the capabilities and limitations of the three potential strike platforms known, the question becomes:

What does a Theater CINC gain by employing an SOV armed with a CAV in an operational level combat scenario?   This question will be answered by comparing the SOV/CAV with two currently available strike packages, a Navy CVBG and an Air Force AEF.

The three primary areas of interest to the Theater Commander are: How effective are the forces against a target? How long will it take the force to destroy the target? and What are the risks to aircraft and personnel?

This thesis will examine these questions using a scenario against an enemy Command and Control (C^2) Bunker. As seen in the previous section of this chapter, the analysis of the effectiveness of a weapons platform against a heavily defended target is not easy and does not readily lend itself to campaign analysis. To obtain the answers to our questions with all of the probabilistic and dependent factors involved in the scenario, it requires using a simulation.

1.   Previous Simulations

Simulations and wargames such as Generalized Campaign Analysis & Modeling System (GCAMS), Joint Theater Level Simulation System (JTLS), Janus and others are well suited to conduct simulations based on existing weapons systems and scenarios, but consume considerable time and resources to make adjustments. This is not advantageous for the evaluation of weapons systems that do not exist. If the design parameters change as we conduct experiments, we may not be able to incorporate those changes in a reasonable fashion or expeditious manner.

2.   Component Based Simulation

Component Based Simulation provides the best possible solution to answer our questions. Weapons systems, scenarios and other objects can be quickly assembled from ready-made components, using the process of composition. Once one question is

answered, more will invariably arise. Component Based Simulations offer the ability to quickly change the simulation by changing the way elements are composed or making small changes to specific components without throwing the rest of the simulation into disarray. Therefore, this thesis will use the concepts of event driven simulation and component software to conduct our analysis.

Major Arent Arntzen of the Norwegian Air Force developed the first draft of Modkit to take advantage of composite software design so that a planner could gain an effective use of his time to develop and test air defense planning strategies through simulation.   Expanding that architecture enables us to evaluate a new scenario. Given a high value target and an enemy with a significant air defense capability, the question becomes, "What is the best way to attack that target and what are the best platforms to complete the mission?"

A working group under the guidance of Dr. Arnold Buss at the United States Naval Post Graduate School is exploring Loosely Coupled Component (LCC) Software Design including Major Arntzen's work in a number of military applications. The goal is to build a library of components that are functional, well tested, and reliable for a modeler to use in testing multiple theories about air defense systems, defense against anti-ship cruise missiles, naval combat and strike warfare. By building components that simulate single capabilities, such as motion, sensing, etc. many programmers will be able to use composition to build larger and more robust models without locking themselves into large, monolithic, single use systems. This should increase reuse and productivity of modelers who build on top of these basic components, component by component, object by object, and simulation by simulation. This thesis will expand on Major Arntzen's work in developing the application of composite software design and use those practices to analyze the question of the impact of a SOV in strike warfare.

II.   PURPOSE AND SCOPE

This thesis has two main objectives. First, examine a revolutionary new weapons system in the SOV/CAV. Secondly, further expand the state of the art of Component Simulation to enable the analyst to quickly compile scenarios for the examination of strategy, or TTPs for the weapons system or changes in system's performance.

A.   PURPOSE: USE COMPONENT BASED SOFTWARE TO TEST SOV/CAV CONCEPT

The primary goal of this thesis is to analyze the relative effectiveness of the SOV against the platforms that currently perform strike missions, namely the CVBG and AEF. However, even with the power of Component Simulation, there must be limits to the scope of the analysis. By focusing on the smallest of details, we will stray from the question and the perspective from which it was asked.

B.   SCOPE

The effectiveness of strike platforms is important at the level of an Operational Commander such as the combatant CINC. The effectiveness of these strike platforms will impact the Commander's overall Concept of Operations and Campaign Plan and may contribute to the ultimate success of the battle.

1.   Abstraction

This analysis focuses on the effectiveness of different strike platforms against enemy air defenses and targets. Therefore, the level of detail concerning maneuverability, G-loading, fuel consumption, etc. are not modeled explicitly. These issues are implicit in basic fly out models that then use a Combat Resolution TableCRT to determine outcomes. The CRT and other Referees incorporate other high-resolution analysis in their judgment. Issues such as logistics (fuel consumption) are assumed as available and are outside the scope of this work.

2.   Operational

This analysis is directed at the operational effectiveness of the strike platforms against particular targets that significantly contribute to the overall battle; but, the overall battle is not portrayed in the simulations.

The Measurements Of Effectiveness (MOEs) chosen in this simulation reflect the decision criteria a Theater CINC ask to decide what forces to task for the destruction of enemy targets defended by air defense systems and CAP. Qualitative analysis will be done to consider the effect particular strike platforms will have on the overall campaign and how they would contribute to the culmination point of such scenarios.

3.   Technical Assumptions

Certain assumptions must be made for the purpose of the analysis. The flight dynamics and reliability of aircraft and the SOV are based on either their current known reliability rates or required values. The flight mechanics of all objects are simulated at only their most basic level.

a)   SOV/CAV works

The first and perhaps the most important assumption is that the SOV can perform as described. While private and public efforts have produced spectacular results, we have no guarantee that SSTO or other rocket technology will make regular space flight possible within the ten year time frame envisioned by this work.. Technical challenges may prevent the CAV from being deployed within the time frame evaluated in this work. A cost effective CAV shell may not be able to successfully re-enter the earth's atmosphere and safely deliver its ordnance to hit the target. Guidance systems may not be effective after re-entry. These technical issues are outside the scope of this work. However, the advantage of Component Simulation design is that as those questions are answered or modifications to existing ideas occur, simulations can be quickly adjusted to incorporate them.

b) Basic cost estimates

The cost estimates for existing aircraft are taken as projections for the future and estimates by the Air Force Research Laboratory (AFRL) are used for the SOV/CAV. Cost benefit evaluation is not the main focus of this work and those values provided are only initial values to begin consideration of such issues. A true cost benefit analysis would require far more research into such issues.

c) Stealth excluded

For issues of national security and to prevent potential misconceptions, the Principal Deputy Acquisition Secretary of the Air Force (SAF/AQ, 3 March 1998) has prohibited the use of notional numbers for the analysis of Low Observable (LO, i.e. "stealth") signature aircraft. Any analysis incorporating LO aircraft must use the actual signature values under the appropriate security guidance. This would require running this simulation at secure computing facilities and setting the data accordingly.

d) No Refueling

Refueling aircraft is considered done safely and effectively using support aircraft not involved directly in combat. Any further analysis of this issue should be considered in a logistics oriented simulation.

e) Basic tactics and weapons systems simulation

This analysis is focused on the platforms involved, not on specific weapons systems themselves. There are many other simulations (such as the Joint Anti-Air Model) that can be used to conduct high-resolution models of missiles against aircraft or counter tactics against those weapons. The level of fidelity in this thesis incorporates the results of those values in a CRT but does not duplicate them.

Now that the scope of the question and the methodology to answer it has been addressed, it is time to explore the means by which Modular Simulation is done and expand the existing state of the art to fulfill the goals discussed.

C.   EXPANDING COMPONENT BASED SOFTWARE TO ENABLE THE TESTING OF NEW CONCEPTS

Most current military simulations are big, monolithic, and require large staffs to get even the most simple training or tactical scenarios running. This detracts from the analysts' ability to collect data and gain insight into the problems placed before them (DMSO 1995).   Further, delays caused by this inflexibility prevent analysts from performing excursions of the simulation, conducting sensitivity analysis and gaining insight into the effects of changes. This prevents effective use of simulation to analyze proposed weapons systems. If knowledge is gained in the laboratory or on the test range, it would be beneficial to incorporate those changes in a model and gain insight into whether or not the project should continue or directions further developments should take. This is the goal of STEP.

1.   Build military units from components

The secondary goal of this thesis is to expand the state of the art of Component Simulation and enable the analysts to use existing libraries of components to build military units and exploit the flexibility of component design to make changes as needed.

2.   Not the full arsenal

To illustrate the effectiveness of Component Simulation design, components will be designed to model the actions and effectiveness of strike aircraft and the proposed SOV against a heavily defended target. A full list of all military units in the world's inventory for the programs used does not currently exist. In fact, it can be strongly argued that a monolithic Order Of Battle (OOB)

18

as such is neither desirable nor even possible to create with the rapid change in weapons technology and political events.

3.   Others can be made in future

Following the example of the author and using the examples provided in the Annexes of this thesis, other analysts can build their own objects and scenarios to analyze their own military questions. The analyst should feel free to build his own designs and interactions without fear of "breaking" the existing system or corrupting a database.

III.   CURRENT DESIGN OF MODKIT FOR SIMULATION

In this chapter we will explore the concept of software components for simulation and discuss the current mechanisms that make component simulation work.   First we will examine the different mechanisms for handling time then the ability to compose objects from sub-components and then the programming mechanism to allow information to be transferred from one object to another.

A.   DISCRETE EVENT DRIVEN VERSUS FIXED TIME STEP SIMULATIONS

In constructing a simulation, the modeler must decide on a time advancement mechanism. For military simulations, this is a critical factor. How a unit moves, and what it attacks and the synchronization of its actions are dependent on how the passage of time is handled. There are two ways of handling the advancement of time in a simulation: Time Step and Event Step.

In a Fixed Time Step simulation, the passage of time is modeled with discrete time steps of turns of length Δt. All actions occur during the length of that time. For example, an operational level simulation such as TACWAR may run with a Δt of half a day. This increment of time is generally associated with the smallest cycle time of the smallest component of interest. Daylight and night occur every other Δt and units move a distance equal to how far they can move in twelve hours during daytime or nighttime conditions.

In a Fixed Time Step simulation, the status of all units is updated after each turn. At the tactical level, a high fidelity model may represent an individual soldier with a much shorter Δt while at the Brigade level we are concerned with a company in half day increments. At each Δt, the system must poll each unit and compute its current status. This can consume computation time for units doing activities that currently have no impact on other units. An additional complication for Fixed Time Step simulation is determining how to handle events that occur at intervals smaller than Δt. For instance a

fighter making an attack on a tank in an operational level simulation would reduce the tank count of a brigade by one tank. If that fighter can make six attacks during a twelve hour period, Δt needs to be reduced to two hour steps or just sum the attacks and apply them at the end of the current "turn" which likely induces errors into the model.

A major difficulty for Fixed Time Step simulations is the manner in which the length of Δt can affect the outcome. If a tank has the ability to acquire and attack a target faster than its opponent, the simulation results will only show that result if the difference between their engagement times is less than Δt. Otherwise, we may never see that ability affect the outcome (Warhola 1997).

In Discrete Event Driven simulation, time is modeled using an Event List. Each element of the simulation is capable of entering an event onto the list based on its own calculations. For example, our fighter schedules a take off event and an arrival event at its destination. Rather than query the fighter at every Δt as to its location, the simulation only queries when an event needs the information. At that moment, the fighter can report its position using dead reckoning (e.g., find the current time, subtract the time it departed and use the resulting difference to determine its current location). In this way, each unit can schedule when events occur according to the needs without requiring the whole simulation to update its status.

A simulation system called Simkit employs the Discrete Event Driven Simulation process using a JAVA based program (Stork, 1996). By maintaining a Master Event List in a Schedule object, Simkit keeps track of time and notifies components when events are to occur. Objects in the simulation place events on the Event List based on the time they are to occur. These same objects can then remove these events if other events cause them to require cancellation. For example, a tank's scheduled acquisition event may be canceled if it is destroyed first. The Schedule would then remove that Event from the Event List.

B.   COMPONENT SOFTWARE DESIGN

Component Software Design uses the concept of composition. Complex entities are modeled by breaking them down into their constituent aspects or characteristics. One

component might be responsible for the color of an object while another keeps track of its location, etc. The goal is to not only ensure that each important aspect is modeled but that the aspect is available to the rest of the object when required. A mechanism is therefore required for an object to obtain its properties from its associated components as required.

The origins of Component Software Design for Simulation came to fruition at the Naval Postgraduate School in Monterey, California when a Norwegian Air Force pilot (Major Arntzen, 1998) brought together a simulation program with the concept of loosely coupled components and developed a communication framework.

1.   Loosely Coupled Components

At the same time these simulation processes were being explored, Professors Buss and Bradley were developing the use of Loosely Coupled Components for military applications.   A component is fully self sufficient on its own. It requires no outside influence to operate and may indeed have no idea other components even exist. Each component does its own particular task. In the example of a Fire Brigade, one component fills buckets of water extremely well, the next carries buckets extremely well, and the last throws buckets of water with great efficiency onto fires. Each member of this brigade has no idea about the other until they are placed in communications with them. When a fire breaks out, each goes to work in his/her own fashion until the fire is extinguished, not because a large fixed object was set to the task, but because of the grouping of small components. If a new bucket carrying object was created that could carry two buckets instead of one, we can plug the new object into the scenario without altering the other two components (Bradley 1998). The advantages of this design give us the ability to re-use components, increase productivity, and enable the user to build high quality components without getting lost in the complexity of a large program.

2.   Component Software for simulation

In his thesis, Major Arntzen developed the concept of software components for simulation and introduced the requirements for the components along with the advantages of their implementation (Arntzen, 1998). Each component must be able to stand on its own. We ensure they are composeable by ensuring that each component is able to

communicate with other components using a strict syntactic process. This ability to create composites of loosely coupled components, each with their own assignments and tasks, is a very powerful tool. It enables the programmer to break down problems into manageable pieces and yet remain confident that each component will be able to contribute its part to the solution.

Additionally, as each smaller component is developed and tested, it is available to the programmer for future compositions. This means that after each component is completed and tested, the author can encapsulate it (along with sufficient documentation) and place it on the shelf for another programmer to use. The next programmer can then pull the component off the shelf and use it for his next task confident that the errors he may encounter are in his own work and not have to second guess the functionality of the component being used.

Major Arntzen's work on Modkit laid the foundation for the component architecture with the BasicModComponent as the most basic simulation component that meets all of the requirements to ensure composition. A further development is ModContainer, a managing shell into which components can be put. Each component within the ModContainer brings its own properties and Events; but, to the outside world, all of the actions and properties appear to be the doing of the ModContainter shell.

These components, the BasicModComponent and the ModContainer are the building blocks of the objects within the simulation. However, as Modkit and the concept of Component Software evolves, so will the components used. A part of that evolutionary process is the specialization of the components.

C.   INFORMATION DISSEMINATION

Major Arntzen constructed the basic foundations for simulation component software in the original design of ModKit. Here we will explore the mechanisms used to enable these Loosely Coupled Components to communicate with each other, which is the basis for composition.

1. SimEvents

Major Arntzen modified his ModComponent to extend Professor Buss' SimEntityBase enabling it to be used in a Discrete Event Driven simulation process. The Schedule class acts as the Time Master to keep track of events as they occur.

The Schedule/Time Master removes the next occuring event, advances the clock to the scheduled time of that event, and informs the object that placed the event on the list that its event is ready to be processed. The time is the main property of the Time Master while the announcement of an event from the Event List is its main event function. Currently, Modkit uses the Schedule class Simkit to act as the Time Master and to maintain the Event list. The Schedule class is a static and singular object that never has to be instantiated and can be called by any object at any time. Because only one exists, we know what object and method to call upon during the simulation without having to maintain a reference to it.

Simkit, and by extension Modkit, currently uses objects of type SimEvent on the Event List. When executed, a SimEventcauses object that scheduled the event execute a corresponding appropriate method when it occurs. In this manner, the SimEvent message is used in Modkit for directional communication from the Time Master to the *specific* component.

2. ModEvents

ModEvents are designed in a manner that all components can receive and attempt to handle them because of the fixed syntactic interface design built into Modkit. When a component receives a ModEvent message, it sees if it has a method to handle it. It can then broadcast the event to its sub-components and they can turn broadcast it to their sub-components. This is far different from the SimEvent. The SimEvent goes to the specific component that placed it on the Event List or the originator to which it is directed. SimEvents may be broadcast to SimEventListeners in a manner similar to ModEvents, but, a SimEvent is not re-broadcast further as is a ModEvent. Therefore, ModEvents are best used for inter-component communications while SimEvents are best used from the Time Master (Schedule) to the sub-component that placed it on the event list.

3. Property Settings

Modkit's architecture currently allows an outside source to serve as a "property source" for another component. When the component receives a "set" or "get" property command ("accessor method"), it finds the first component holding that property and records the change or reports the current state. In order to be accessible in this manner the accessor methods within the component handling that property must be public; but, the ModComponent Architecture that uses this finds the right place to set the property on the fly during the simulation.

IV.   DEVELOPMENT OF MODKIT DESIGN FOR COMBAT SIMULATION

This chapter describes the new components and procedures employed in the Combat Modeling Simulation package to simulate military applications for existing and proposed weapons systems. These components expand the current state of the art of Component Software Design for simulation.

Major Arntzen's work in ModKit left us with the BasicModSimComponent, a self-standing component that can be used with others in composition to build more sophisticated simulations while being able to place events on the Event List. The BasicModSimComponent can be likened to a generic cell, a simple self-sustaining organism able to be adapted for those roles required in the simulation. The BasicModSimComponent is the first step in a line of evolutionary developments for ModKit and Component Software Design for Simulation.

ModComponent
starting "cell"

Referees
"Measuring"
Intereaction

Players
"Measured"
Isolated Behavior
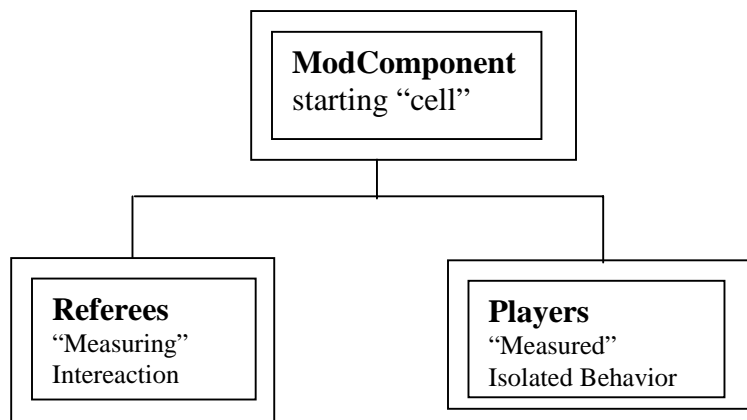
Figure 1. ModComponent Evolution

Components become more specialized in each step of the evolutionary development. Like a cell expressing itself as a red blood cell or a neuron, components retain all of the elements of the original, but become more specialized to accomplish specific tasks.

Figure 1 illustrates the immediate branching within this evolutionary tree. It is the difference between the objects of interest and the objects that adjudicate the interactions between them. The simulation is built to model the interaction between objects; but, the final measurements of the experiment are done on the objects themselves. To understand the results of our simulation we measure the "physical" components status and not the adjudicating components. Like a Wargame, the final goal is to see which side won and perhaps why. In a Wargame a Referee's task is to adjudicate the outcome of an interaction of two objects. If a chart has to be referenced, a die rolled, or some other method used, the Referee does it and not the objects themselves. In our simulation models Referees will play similar roles between objects that represent physical entities ("Players"), since the objects themselves should not unilaterally adjudicate outcomes. As **Figure 1** indicates, Referees are a different branch of the evolutionary tree than the Players.

Placing these responsibilities with an Umpire or Referee has certain advantages. First, we can change the manner in which we think objects interact without having to go in and change the objects themselves. This means that if we wish to change the level of detail of the interaction we can use one kind of Referee one time and another at some later date, perhaps changing even in the middle of the simulation. Second, the Referee can be designed to safeguard its process to prevent Players from unwittingly adjudicating outcomes improperly, since a Referee will be more impartial than the Players. Third, this design enables the designer to be free to place only the attributes of the object in it and take a minimalist attitude. This design will enable the designer to simplify the work.

A.   MODELING COMBAT ENTITIES

While the Referees address the interactions between components, the "physical" components can focus on their own specific characteristics and internal states. By having a clear distinction between what an object is and how it interacts with the outside world, we are empowered to build objects in a manner to deal with only with those characteristics that are instrinsic to the nature of what they are modeling. We can leave the issue of their interaction with other objects to Referees. This design format frees us to

make changes to either the object's state or its interaction with others, without requiring us to change both. We can change the state of the physical object or we can change the Referee/Mediator used to interact between objects, but not have to change both.

The goal of this section is to refine the work that Major Arntzen began and to extend the process he started by defining the objects that we are going to use in our simulations. These components build on and extend those developed in Simkit and Modkit (Arntzen, 1988).

The first step in continuing this work was to break the original components down into their most basic constituent elements. By ensuring the most basic and fundamental tasks are broken down and delegated to the smallest component, we can test that component and ensure it does that one task extremely well. This ensures reliability of that component, enables the author to "shrink wrap" the component and allows the user to use it "out of the box" without altering the component or worrying about its ability to function properly. The author can focus on his current work and focus his energies (and trouble-shooting) on the current component when larger components are being built and use these smaller components. Additionally, the author can use the functionality of the basic component to do the work of his new object as a delegate to keep track of the object's state.

```
┌─────────────────────────────────────────┐
│ BasicModComponent                        │
│ Basic "Cell"                             │
│ Holds properties and handles ModEvents   │
│                                          │
│                                          │
└─────────────────────────────────────────┘
                    │
┌─────────────────────────────────────────┐
│ BasicModSimComponent                     │
│ SimEntityBase                            │
│ Handles ModEvents from Schedule          │
│                                          │
│                                          │
└─────────────────────────────────────────┘
                    │
┌─────────────────────────────────────────┐
│ BasicCombatEntity                        │
│ Additional Properties: Side EntityType,  │
│ Force Level, Alive, Parent.              │
│                                          │
│                                          │
└─────────────────────────────────────────┘
                    │
        ┌───────────┴───────────────┐
┌──────────────────────┐   ┌──────────────────────────┐
│ BasicMover           │   │ CookieCutterSensor       │
│ Hold properties of   │   │ Hold properties of       │
│ Velocity, Moving,    │   │ maximum range            │
│ Speed.               │   │ Handles Detection Events │
│ Schedules arrivals,  │   │                          │
│ departures, stops.   │   │                          │
└──────────────────────┘   └──────────────────────────┘
```

Figure 2. Combat Entities

We saw the first branching of the evolutionary process of components in the last chapter. The split was between components that represent the "physical" objects in the simulation and those that model the interaction between them, literally between the measured and the measuring. Within the realm of the "physical" components, we can see further specialization of components while maintaining generalization. The tree is branching but we have not reached the outer branches and leaves just yet. Particular simulations may need a basic object that conforms to a common and fundamental aspect of all the objects being modeled. We need baseball players for a baseball game and combat entities for a combat model.

Here we will explore the most fundamental nature of the objects we are modeling and use those characteristics as the basis for all "physical" objects in the simulation.

Combat simulations are designed to portray military units so we need to specialize Modkit's BasicModSimComponent and ModSimContainer to incorporate characteristics of military objects. The BasicCombatEntity models the most basic component in the combat simulations of this thesis. Figure 2 illustrates the extension of the BasicModSimComponent through to the BasicCombatEntity. The BasicCombatEntity has basic characteristics of what side or "color" it is, what kind of object it is (how it would be perceived by other units), its force level or "health", and whether or not it remains alive or functional. Each BasicCombatEntity has five built in properties: Side, EntityType, ForceLeve, Alive status and Parent. Each property represents an aspect of a combat object in a conflict simulation.

1. Side

Side is the characteristic of what side a component is on. Common NATO color distinctions are used to represent units for wargames and maneuvers. Rather than using an integer or string value to represent side, a fixed set of "colors" is available in the standard Size class. This avoids the fragility of strings and integers and gives us a tight control on what those Sides are. The colors are Red, Blue, Orange, Purple, Green, White (usually used for neutral) and Unknown for (used when an object's origin is not known). Sides can be compared to see if they are the same or queried for a String appropriate to its color. Side is often used for Identification Friend or Foe (IFF) steps in the Detect to Engage (DTE) sequence. The contract we have with all components is that they are on a side. The default side of "Unknown" is used if none is specified.

2. EntityType

Type is the characteristic of how the object appears to others and should give a brief description of the visual aspect of the "real" world object being modeled. An object may have EntityType such as "Tank" or "Jet," or perhaps be more specific. While the EntityType characteristic is strongly controlled, it does not have a fixed set of potential forms. The set of potential EntityType is known at the beginning of a simulation and each object has an EntityType property. An EntityType must therefore be established before it can be set as a property on an object. EntityTypes can be compared to each other

for equality. The EntityType characteristic is often used for the weapons assignment/response portion of the DTE sequence.

3. Force Level

As CombatEntities attack each other, they can degrade each other's "strength" or "health" or other measurement of capability. As the unit takes damage, the Force Level is used to record Fractional Damage. Whether a tank battalion loses a portion of its strength or a ship loses a portion of its hull integrity, the forceLevel otherwise can be used to measure these factors as a percentage of the unit's starting value. ForceLevel can then be affected in combat by having a Referee (most likely the Umpire) adjudicate an interaction with a hostile unit. Units can start off under-strengthed or reinforced; but, in general, they should start out with a forceLevel characteristic equal to the default level of 1.0. ForceLevel can be adjusted in combat and queried returning a double value number, usually between 0.0 and 1.0. ForceLevel can be used to simulate the combat potential of a unit and/or its degradation by other units.

4. Alive

As BasicCombatEntity attack each other, some may be damaged to the point they are no longer viable. The Alive characteristic is a boolean that usually starts off true, but may become false when the unit is killed. The BasicCombatEntity has a default method for a forceLevel set equal to or less than 0.0. In this case the BasicCombatEntity is automatically killed, sets its Alive characteristic to false and reports its death in a ModEvent. The BasicCombatEntity can be queried as to its Alive status which will be returned as a boolean. The Alive characteristic can be used to indicate whether an object is capable of action or is inert.

When an object is considered to be dead, it cancels all events that it has scheduled. Other objects, especially Mediators will generally disconnect from dead components to save cycles and memory.

5. Parent

A BasicCombatEntity may be part of a larger object that can be destroyed as a composite object. Generally, when the object that contains others is destroyed, the

objects within are destroyed. The BasicCombatEntity is designed to listen for deathEvent messages from their parent objects in which case they self destroy.

B.   MODELING MOTION

The simulation of an object's motion through space and the ability of sensors to detect an object are critical in the simulation of Air Defense systems and strike platforms. The next two component types we will explore deal with the modeling of motion and sensing. Within these categories we will explore some refinements of previous work and their extension into more advance models.

To simulate motion, a coordinate system is required. Modkit employs a set of Coordinate classes to do this. A Cartesian coordinate system maintains the three normal axes; X, Y, & Z and combines them with a time factor T.   T represents the time when the object was or will be at the Cartesian point. These Coordinate classes incorporate functions we'll need for calculations such as determining the distance between them, addition, subtraction, scalar multiplication, etc.

1.   BasicMover

Using our coordinate framework we need to have an object start in a location and move to another at a given speed. This is accomplished with the BasicMover class. The BasicMover is designed to simulate an object that moves from one location to another with a smooth linear speed or set in a particular direction with a set speed. The BasicMover keeps track of where it is, its destination, an arrival time, velocity, and current speed.   The BasicMover was one of the first objects developed by the working group. The current BasicMover was developed from the original code written by Oray Kulac (Kulac 1999).

When the BasicMover's destination is set, it will set its velocity and speed accordingly and schedule its arrival time. If queried about its current location, the BasicMover will calculate its current position based on the time it departed and its current velocity using very simple dead reckoning. There is no need to time-step the motion process, only to schedule an arrival time. If the BasicMover has its velocity, set it has no destination or arrival time and does not require scheduling of an arrival event.

The BasicMover maintains it's the data of its motion, location etc. and informs the outside world of its change of state through the syntactic process of the BasicModEvent. A VelocityChangedEvent is fired whenever the BasicMover's velocity is changed by either having a destination set or a velocity set. An ArrivalAtLocationEvent is fired whenever the BasicMover arrives at the last destination it was given. A MoverStoppedEvent is generated whenever the BasicMover stops moving. A MovingViolationEvent is generated when the BasicMover is given a destination that it cannot reach either because of its maximum speed or the time is impossible. These Events enable the BasicMover to interact with the outside world to keep track of where it is and its velocity.

2.   SmoothLinearMover

The SmoothLinearMover is used to simulate an object moving along a set route of points in the Cartesian system, without stopping at each point. It will have linear motion along each leg. Other components could use the SmoothLinearMover to approximate non-linear motion by breaking a path down into a series of points on which uniform linear motion is used.

The SmoothLinearMover uses the BasicMover as its delegate to keep track of its current location, velocity, etc. The SmoothLinearMover pulls follow on destinations from a queue or list of destinations whenever it receives an ArrivalAtLocationEvent from its BasicMover. When the queue is empty the SmoothLinearMover stops and generates a MoverStoppedEvent.

The SmoothLinearMover does not allow the BasicMover's ArrivalAtLocationEvent to propagate so that no other object hears that event. All of the other events the BasicMover generates are changed to make the SmoothLinearMover the source. The outside world perceives the SmoothLinearMover as doing the movement and supplying the current state of the object while the BasicMover does the work. This enables interested objects in the simulation to observe the SmoothLinearMover in motion without having to see it start and stop as it goes along. Like the swan gliding across the pond, we don't see its webbed feet furiously beating underneath the surface of the water.

3.   PursuitMover

The PursuitMover uses the BasicMover in the same manner as the SmoothLinearMover to keep track of where it is. Its destination is determined by the target it is set to pursue. It heads directly for the point where the target is currently. It updates that position at a set interval. This results in a tail chase towards the target. This can be used to model objects that try to catch another object in this manner.

4.   InterceptorMover

The InterceptorMover extends the PursuitMover. Its destination is set to a point in front of the target based on the target's velocity. This can be used to model objects that move in an intercept course to catch their target.

C.   MODELING SENSORS

In a Fixed Time Step simulation, sensors must be queried every Δt, as to whether or not they see a target. A better method for Discrete Event Driven simulations is to determine when an object will be within the detection geometry of the sensor and schedule that detection accordingly. Rather than have the sensor do the work, Modkit uses a set of Mediators acting as a third party that examine the motion of the target and the sensor and schedule the detection time accordingly. Because the Mediator does the "thinking", the sensor becomes a placeholder, maintaining its own state and supplying information to the Mediator as required. The sensor is only required to provide data that describes the geometry of its detection pattern and any other pertinent information. The sensor must also be able to have a detection set upon it with the data of the target. Even then, the sensor is an empty shell that only passes the detection event to anyone listening.

The sensor fulfills our requirement that it do the most basic fundamental tasks required. In order to determine when detection will occur, the mediator needs to compute the relative velocity of the target and sensor. The sensor will require a locator, generally this will be either a BasicMover or SmoothLinearMover, but it can be any component that keeps track of locations and velocities.

Most methods of detection involve the use of either electro-magnetic radiation or sound. Those forms of energy generally disperse in an inverse range squared relation.

That is the intensity of the energy at a distance from its source is usually at a fraction in proportion to the square of the range. True, highly detailed simulations focusing on detection methods may want to look at this process; but, the objects used to simulate sensor detection are based on linear models. The range that an object can be detected is a linear product of the sensor's sensitivity and the characteristic affecting detection. This process has been used successfully by wargames such as Harpoon4 (Bons & Carlson, 1996) and military technical planning guides such as the Threat Weapons Guide (Department of the Air Force, 1997).

Sensors of interest in include the standard Cookie Cutter Sensor, the Cookie Cutter Altitude Sensor, the Cylinder Sensor (a combination of the two), the Cookie Cutter Cylinder Sensor, the Cookie Cutter 3D Sensor, the Active Sensor, the Passive Sensor, the Emissions Sensor, and the Fire Control Sensor. We can use these to model most existing military sensors in a combat simulation.

1.   Cookie Cutter Sensor

The CookieCutterSensor models a component geometry and detection value. The CookieCutter2DSensor describes its maximum range on the X and Y axes of the Cartesian system while a CookieCutter3DSensor describes its maximum range on the X, Y, and Z axes.

The CookieCutterSensor describes basic range geometry and the Mediator uses it to determine when detection occurs. When the detection occurs, the Mediator sets the detection on the sensor with the target and the sensor informs other components that detection has occured.

2.   Cookie Cutter Altitude Sensor

The CookieCutterAltitudeSensor maintains the altitude band of the sensor in the same vein as the CookieCutterSensor. The mediator uses this information to tell the sensor when the target is within this band.

3.   Cylinder Sensor

By combining the roles of the CookieCutterSensor and the AltitudeSensor we can generate a sensor that maintains a geometry of a cylinder with range on two axes and an

altitude band on another. We can make a similar combination using a three dimensional range with an altitude band to create a sphere with a plane cutting through at a particular level of an axis.

4. Cookie Cutter Cylinder Sensor

The CookieCutterCylinderSensor extends the idea of the CylinderSensor by maintaining a three dimensional range and an altitude band on the Z-axis. It has a floor and a ceiling between which the target must be to be detected. We also add a boolean to track if the sensor is enabled. The sensor cannot detect anything if it is not enabled.

5. Cookie Cutter 3D Sensor

The CookieCutter3DSensor is used to simulate basic weapons envelopes, visual sensors, decision ranges, terrain, etc. By combining the Cylinder Sensor with a Line Of Sight (LOS) factor, we can simulate sensors operating on the surface of a globe.

6. Active Sensor

The ActiveSensor extends the idea of the CookieCutter3DSensor but changes the geometry of the range. Instead of having a fixed maximum range, the ActiveSensor has a value β. β is the maximum range at which the ActiveSensor can detect a target with a RadarCrossSection (RCS) of 1 square meter. Its own RCS delegate maintains the RCS of any targets. The Mediator multiplies β by RCS and that becomes our new maximum range.

The ActiveSensor also has a value of α. When a target is within the geometry of the ActiveSensor it is not automatically detected. Instead the time of detection is then scheduled. The higher α, the sooner the target will be detected and the lower α, the later the target will be detected. Regardless of when the time of detection is scheduled the target is no longer detected when the target leaves the geometry. In addition to tracking whether or not the ActiveSensor is enabled, it also maintains a state of being active or not. The ActiveSensor cannot detect any targets if is not active.

The time until detection occurs is $T_d = - \alpha * \ln(1-u)$, where u is a uniform (0, 1) random number. This formulation comes from the inverse of the continuous looking sensor model.

The ActiveSensor is used to simulate active sensors that can be counter detected. It is primarily used to simulate radar units, with an α based on the rate contacts transition from radar blips to actual tracks, while the β is based on the maximum range the radar can detect a 1 meter squared target. The maximum range of detection goes up linearly to the RCS of the target. The Floor and Ceiling are based on the minimum and maximum detection altitudes the sensor is capable of finding a target at.

7. Passive Sensor

The PassiveSensor is very similar to the Active sensor and extends the ideas of the CookieCutter3Dsensor. The PassiveSensor is concerned with the Infra-Red Signature (IRS) of the target. The PassiveSensor has a value of ϕ in place of β. ϕ is the maximum range at which the sensor can detect a target with an IRS of intensity one and goes up linearly with the intensity. Otherwise, the PassiveSensor has a value of α used in the same manner as the ActiveSensor.

The PassiveSensor is used to simulate passive sensors that detect the heat of another object. It is used primarily to simulate Infra-Red sensors, Forward Looking Infra-Red Radar (FLIR), etc.

8. Emmisions Sensor

The EmmisionsSensor is very similar to the ActiveSensor but is designed to detect the ActiveSensor. The EmmisionsSensor maintains and uses α in the same manner as the ActiveSensor but has a value called γ in place of β. γ is a multiple that describes the range an EmmisionsSensor can detect an ActiveSensor. The maximum range of detection is equal to γ multiplied by the target sensor's β multiplied by 10. Therefore, it is a multiple that is applied to the maximum range at which the target ActiveSensor can detect a target with an RCS of 10. The target ActiveSensor, cannot be detected by the EmmisionsSensor if the ActiveSensor is not active.

The EmmisionsSensor is used to simulate electro-magnetic listening systems designed to counter detect radar systems. These include aircraft Radar Acquisition Warning (RAW) systems and other systems designed to warn or detect operators of active radar is in use.

9. Fire Control Sensor

The FireControlSensor extends the ideas of the ActiveSensor, but is designed to find and track one specific target. Rather than survey its geometry, it attempts to find a specific target it has been assigned, usually by another surveillance sensor. In addition to the values maintained by the ActiveSensor it maintains how many channels it can keep operating and how many are currently in use. When the number in use is equal to the number it can operate, it is considered overloaded and cannot add anymore tracks. The FireControlSensor is used to simulate active radar fire control systems such as those used on Surface to Air missile systems.

D. MODELING DETECTION

Because the Sensor is essentially a shell that describes the geometry of the sensor and other data, no "work" is done in the sensor. A Mediator that examines the relative motion of the target and sensor does the real work and determines when the target will be within the geometry of the sensor and when it will be detected.

This work can be done either by the mediator itself or by employing SubMediators. A Mediator can break down the computational work of scheduling a detection and un-detection by dividing up the work. The Mediator then only has to do a logic test each time a SubMediator determines its part of the calculations have been met.

Figure 3. Use of Mediators and Sub-Mediators.

In the case of the CookieCutter3DSensorMoverMediator, which mediates between a CookieCutter3DSensor and the target, the task is divided into three parts: Range, Altitude and horizon LOS. The main Mediator generates three Sub-Mediators. The first is a RangeSubMeditor that consults the sensor's maximum range and determines when the target will be within that maximum range geometry and schedules the event. When the event is executed, it trips the range geometry on the master mediator. The AltitudeSubMediator consults the sensor's floor and ceiling and determines when the target will be within the altitude band of the sensor and schedules the event. When the event is executed, it trips the altitude geometry on the master Mediator. The final SubMediator is the LOSSubMediator designed to check horizon LOS requirements, based on the two objects X, & Y position and height. When the two objects have line of sight on each other, the LOS logic is tripped on the Master Mediator.

When any logic check is tripped, the Master Mediator checks to see if all three are true., in which case detection occurs. On the other hand, if the target is within the geometry, but one of the SubMediators then trips the logic to false, the target will no longer be detected and an un-detection is executed. This ensures that we properly model the interaction of the sensor and target. All three conditions (range, LOS and altitude) must be true for a detection to be possible. If just one is not, no detection can occur.

Each sensor Mediator is designed to mediate between one sensor and one target. Each sensor-target pair is assigned a Mediator based on the characteristics of the sensor. When either the target or the sensor is destroyed, the Mediator disconnects and no longer listens to either object. Additionally, the Mediators are generally disconnected from the sensor and target when the current run is completed.

Table 1 is a list of Sensor Mediator pairs used:

CookieCutterSensor	CookieCutterSensorMoverMediator
CookieCutterAltitudeSensor	CookieCutterAltitudeSensorMoverMediator
CookieCutter3DSensor	CookieCutter3DSensorMoverMediator
ActiveSensor	ActiveSensorMoverMediator
	ActiveRangeSubMediator
	AltitudeSubMediator
	LOSSubMediator (Line of Sight)
PassiveSensor	PassiveSensorMoverMediator
	PassiveRangeSubMediator
	AltitudeSubMediator
	LOSSubMediator (Line of Sight)
EmissionsSensor	EmisssionsSensorMoverMediator
	EmisssionsRangeSubMediator
	AltitudeSubMediator
	LOSSubMediator (Line of Sight)
FireControlSensor	FireControlSensorMoverMediator
	ActiveRangeSubMediator
	AltitudeSubMediator
	LOSSubMediator (Line of Sight)

Table 1. Sensor/Mediator Pairs

The FireControlSensor employs a special kind of mediator. Instead of a mediator being built to mediate between the FireControlSensor and the target at the beginning of each simulation run, it is generated only when need. A FireControlSensor generates its own mediator when assigned a target. The FireControlSensorMoverMediator then completes its task and tells the FireControlSensor when it detects the target. When the FireControlSensor loses track of the target, because either is destroyed or the target leaves the maximum range of the FireControlSensor, the mediator disconnects and no longer

listens to either the sensor or the target. If the FireControlSensor is reassigned to track the target, a new FireControlSensorMoverMediator will be generated.

E.   MODELING BASIC RESPONSES AND LOGIC BEHAVIOR

The manner in which objects respond to a threat needs to be modeled in most simulations. While there is much discussion these days regarding Artificial Intelligence (AI), there is no model currently available that can be used to model a human response in most circumstances with which we are concerned. However, basic responses to particular situations are often simulated using simple rules.

The decision making process of an organization or system can be broken down into component pieces just as the tasks facing our Mediators were able to be broken down for SubMediators to handle. Each component in a chain of decision making chain can complete a simple algorithm that represents its "decision."
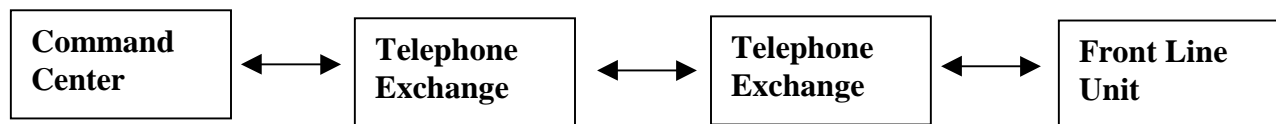
Figure 4. Basic Behavior Chain

Additionally, the transfer of orders from one object to another can be simulated as well. Imagine a series of telephone exchanges from a command center to a military unit as shown in Figure 4. Using ModKit we can simulate each of those components in the communications grid. The command center generates a command and sends it to the military unit via the telephone exchanges. The command center, the exchanges and the military unit can be simulated as components. The telephone exchanges may only have the task of repeating the message down the line by receiving a ModEvent and re-transmitting it. We can even introduce some sort of delay factors for the messages before they continue on. As the message is being sent, an enemy force may interdict one of the

exchanges, thus preventing the message from getting through the network. This example demonstrates one simple way to simulate command and control issues for military units.

To evaluate strike platforms against defended targets we must model the defensive systems used. In order to simulate the way Surface to Air Missile (SAM) systems operate we must simulate the steps or events in the Detection To Engagement (DTE) sequence:

a) Detection
b) Tracking
c) Identification
d) Weapon Assignment
e) Prioritization
f) Engagement
g) Evaluation
h) Re-Engagement
i) Evaluation

At each step there is an automated system, person or both making decisions about how to respond and whether or not to continue the sequence. This sequence, like many other military responses, has been studied and deliberated extensively. Doctrine has also been generated for military personnel to use as a guide for their actions. While we may not be able to use AI to fully simulate how personnel will respond to certain events, we can use published doctrine as our assumed behavior. Further, we can then manipulate that doctrine to determine optimum behavior.

The Combat Modeling Simulation package uses a collection of components to emulate the behavior of systems used in the DTE sequence listed above. The chain of events that goes into engaging a target is broken down into components that execute the basic decision tree of most engagement sequences. Table 2 shows the events in the DTE sequence and what components handle those events.

Step	Component modeling the step
a) Detection	Sensor
b) Tracking	TrackFileKeeper/Correlator
c) Identification	TrackSuperVisor
d) Weapon Assignment	WeaponsCoordinator
e) Prioritization	EngagementQueue
f) Engagement	EngagementCoordinator/Launcher
g) Evaluation	EngagementCoordinator
h) Re-Engagement	EngagementCoordinator/Launcher
i) Evaluation	EngagementCoordinator

Table 2 . Components modeling steps in DTE sequence.

1.   Track File Keeper

The Sensor described in the previous section is a shell that informs the Mediator of its characteristics and accepts detections of targets as they occur. However, the sensor does not keep track of what it detects. It only reports what it detects and un-detects. Therefore, we need a component to keep track of what the Sensor is tracking. The TrackFileKeeper maintains a list of targets that its sensor is tracking. When the sensor reports a detection, the TrackFileKeeper adds the target to its list. The TrackFileKeeper then promulgates the DetectionEvent message to its listeners. The TrackFileKeeper removes the target from its list and promulgates an Un-DetectionEvent message when the sensor reports an un-detection of a target. If another object asks if the TrackFileKeeper still holds a target, the TrackFileKeeper checks its list and reports accordingly.

2.   Track Correlator

Suppose we have a number of passive Infra-Red sensors deployed and we need to get a fire control solution on a target from them. Most passive systems only provide a bearing to a target; but a fire control solution requires bearing and range. We can obtain bearing and range if we track the target with two or more passive sensors. To simulate this requirement, the TrackFileKeeper was extended to create the TrackCorrelator. This

component keeps a list of TrackFileKeepers (or other Correlators) and has a minimum number of sensors/ TrackFileKeepers that must detect the target before it will promulgate the DetectionEvent message. Again the TrackCorrelator will respond whether or not it holds a particular target (with the proper number of sensors) if queried. In our Detection To Engagement (DTE) sequence the sensor initiates the Detection step while the TrackFileKeeper and/or combinations of the TrackCorrelator continue with the Tracking step.

3. Track Supervisor

Once a target has been detected, steps are usually taken to attempt to identify it. Information such as intelligence, IFF, Electronic Emissions, speed, size, etc. are fused together for a decision maker to determine if the target is hostile and needs to be engaged. For the level of simulation being run in this thesis, we will simply ask the target what side it is on. Recall that each BasicCombatEntity has a characteristic of Side. This enables us to simply query the object as needed.

The TrackSupervisor is an extension of the TrackCorrelator. It keeps tabs on the TrackCorrelators/Keepers and listens for a target to be reported. The TrackSupervisor determines the target's Side. When the sensor reports a detection of a target, based on the Side of the target and the current firing policy the TrackSupervisor, will send the detection of the target to the WeaponsCoordinator.

The TrackSupervisor can be more complicated providing greater fidelity by assigning a mediator to randomly generate the time until a determination of what Side a target is from, or even generate spurious Sides for the target. For this thesis, the actual Side will be immediately known. The TrackSupervisor can be used to simulate the Identification portion of the DTE sequence and determine if the DetectionEvent needs to be sent on to the WeaponsCoordinator.

4. Weapons Coordinator

Different targets require different weapons to engage them. Obviously, a missile better engages an aircraft than a torpedo. The WeaponsCoordinator is responsible for responding to appropriate targets with their set weapons. The WeaponsCoordinator will attempt to determine the targets EntityType upon receiving a track from the

TrackSupervisor. The BasicCombatEntity enables the builder to set the EntityType and our TrackSupervisor to determine it. Based on the target's EntityType, we can assign a weapon system to engage it. Directing where the DetectionEvent message is to go next does this. The WeaponsCoordinator can be used to simulate the Weapons Assignment portion of our DTE sequence by directing the DetectionEvent message to the proper weapon's EngagementQueue.

5. Engagement Queue

The position and kinematic behavior of a threat may change the priority in which a weapons system would engage the target. While one threat may be closer, the one behind it may overtake it very quickly. Additionally, a potential threat that is outbound, even though moving faster, may not be considered a threat at all. The EngagementQueue is designed to maintain a list of targets it wants to engage. It evaluates the targets current threat according to a threat criterion (usually closest and fastest closing) and "pops" that target off the list. When the EngagementQueue receives a DetectionEvent message, the EngagementQueue checks to see if the WeaponsCoordinator still holds the target. If it does, the EngagementQueue adds the target to its list. If the WeaponsCoordinator no longer holds the target, the target is forgotten about. The EngagementQueue informs the EngagementCoordinator that it has a target to be engaged after adding the target to its list by forwarding the DetectionEvent message.

The EngagementQueue can be used to simulate the Prioritization step of our DTE sequence. Up to this point every step in the DTE sequence was executed by simply examining the DetectionEvent message at each component along the way and deciding whether or not to forward the message to the next component. We can simulate the interdiction of this logic chain, assuming its components were vulnerable similar to our phone exchange example.

6. Engagement Coordinator

The EngagementCoordinator makes the necessary decisions on when and how to actually engage a target next in line. Its decision process may include firing policies of how many weapons to fire at the target and when to fire the salvoes. The initial salvo against a threat may be one while the final salvo may be two to ensure destruction.

A target may be the closest threat; but, it may not be in range to be engaged. The EngagementCoordinator will have to make that decision and determine how long to wait to re-evaluate the interception range. A Fire Control Sensor may be needed to acquire a solution on the target when it does become time to attack the target. The EngagementCoordinator will assign a FireControlSensor to the target when this time comes. When the FireControlSensor returns a DetectionEvent message, the EngagementCoordinator prepares a MissileEngagementEvent message for the nearest available Launcher. The EngagementCoordinator notes when the first engagement should occur while adding the target to its engaged list. It waits until the initial engagement by the missile Launcher should be completed and sends the target to the EngagementQueue for re-evaluation. If the target is destroyed or no longer a threat, the EngagementCoordinator will no longer hear about it. On the other hand, if the target comes back to the EngagementCoordinator, it will note that it has engaged the target before and use the final salvo size for the next engagement.

If the EngagementCoordinator has multiple launchers or fire control sensors, it will assign the closest available launcher and/or fire control sensor as needed. The EngagementCoordinator can also be used to simulate the Engagement orders and to trigger the Evaluation/Re-evaluation steps of our DTE sequence.

7.   Missile Launcher

Upon receiving a MissileEngagementEvent message from an EngagementCoordinator the MissileLauncher removes the necessary number of missiles from its inventory and prepares its rails, launches the missiles and recovers from launching. While a rail is being used or recovered, it is not available for launching a new missile and will send LauncherBusyEvent message to the EngagementCoordinator. When a rail is ready, it will send a LauncherReadyEvent message to the EngagementCoordinator. When the MissileLauncher is bereft of missiles, it is no longer capable of launching missiles and sends a LauncherBusyEvent message to the EngagementCoordinator.

8. Firing Chain

The Firing Chain is a composite object that incorporates all of the components that model particular agents or systems in a DTE sequence chain. By bringing them together in a composite object we can model the overall actions of a DTE firing chain. We can use this object to model particular systems and their doctrine. We can vary the number of sensors necessary to track a target, use Fire Control Sensors, and multiple launchers in addition to modifying the firing policy of the system. Figure 5 is a flow chart that describes how each object contributes to the overall process of a FiringChain executing the DTE sequence.

The steps in the DTE sequence begin when a Sensor-Target-Mediator sets a detection upon the Sensor. The Sensor now detects the Target and broadcasts a Detection Event with a reference to the Target. The TrackFileKeeper hears that event and re-broadcasts it while adding the target to its detection list for thatSensor. If the Sensor un-detects the Target, the TrackFileKeeper will re-broadcast the un-detection and remove the Target from its list. If we are using passive Sensors that require more than one Sensor to detect the Target to maintain a track, the Firing chain will employ a TrackCorrelator. The TrackCorrelator receives the detection and checks to see if the minimum number of Sensors is tracking the Target by querying their respective TrackFileKeepers. If the minimum number is tracking the Target the TrackCorrelator will propagate the Detection Event.

When the TrackSupervisor receives a Detection Event from a TrackCorrelator (passive Sensors with a minimum number) or a TrackFileKeeper (active Sensor requiring only one), it will check to see if the Target's Side is on its threat list. The TrackSupervisor is modeling the identification step in the process and will only propagate the DetectionEvent if the Target is of the right "color" or "side." If the Target's Side is on the Threat List, it will add the Target to its current track list and propagate the Detection Event.

The WeaponsCoordinator will receive the TrackSupervisor's Detection Event and check to see if the Target's EntityType is on its Entity Type List. The WeaponsCoordinator is modeling the process of matching the proper weapon to the

proper target, missiles for aircraft, torpedoes for submarines, etc. If the Target's EntityType is on the WeaponsCoordinator's EntityType list, it will propagate the Detection Event. The EngagementQueue will hear the WeaponsCoordinator's Detection Event and add the Target to its priority List. The EngagementQueue will propagate the Detection Event to the EngagementCoordinator.

At this point the process is no longer screening the target to ensure that it fits particular criteria, but evaluating whether to generate an EngagementOrderEvent to a launcher to destroy the target. The EngagementCoordinator uses the DetectionEvent as a trigger to evaluate a Target. If the EngagementCoordinator has launchers (and, if required, Fire Control Sensors) available, it will ask the EngagementQueue for the Target with the earliest time of impact. It will then evaluate the current intercept point for that Target based on the closest available launcher. If the intercept point is within the maximum range of the closest Launcher the EngagementCoordinator will set up an engagement. If a Fire Control Sensor is required it will order the nearest available one to do so. When the Fire Control Sensor reports a detection, the EngagementCoordinator generates an EngagementOrderEvent and sends it to the Launcher to launch a weapon at the Target.

When the Launcher gets an engagement order, it will decrement the number of available rails for other missions. If the Launcher has no rails available, is destroyed or is out of missiles, it will generate a LauncherBusyEvent. When a rail becomes available, the Launcher generates a LauncherReadyEvent. When the EngagementCoordinator receives a LauncherBusyEvent it removes the particular Launcher from its list of available Launchers. When the EngagementCoordinator receives a LauncherBusyEvent it adds the particular Launcher to its list of available Launchers.

When the EngagementCoordinator generates an EngagementOrderEvent, it notes when the missile should arrive at the intercept point and schedules a re-evaluation. When the time until the expected interception has expired the EngagementCoordinator sends the Target to the EngagementQueue for re-evaluation. If the Target has been destroyed or has otherwise left the geometry of the Sensors, it will no longer be carried in the EngagementQueue and will not trigger the EngagementCoordinator to respond. If the

Target is still within the geometry of the Sensors and is still considered a threat the EngagementQueue will hand the Target over to the EngagementCoordinator according to its impact time.

When the EngagementCoordinator generates an EngagementOrderEvent from the EngagementQueue, it records the Target on a list of targets it has engaged before. The EngagementCoordinator will tell the Launcher how many missiles to fire at a Target. If the engagement is the first, it will use the initial salvo number, but for targets that have been fired upon previously it will use the final salvo number. This way we can adjust firing policies based on the number of times the target has been fired upon.

The Firing Chain models the system of objects that handle the DTE Sequence. It is a composite object that utilizes each of the components in a organization to support the steps of turning a detection into an engagement order and re-evaluating that engagement. Figure 5 shows how the DTE sequence is executed by the Firing Chain using all of the components described earlier.

The Detection to Engagement sequence is an example of a decision tree that multiple components can be used to simulate.
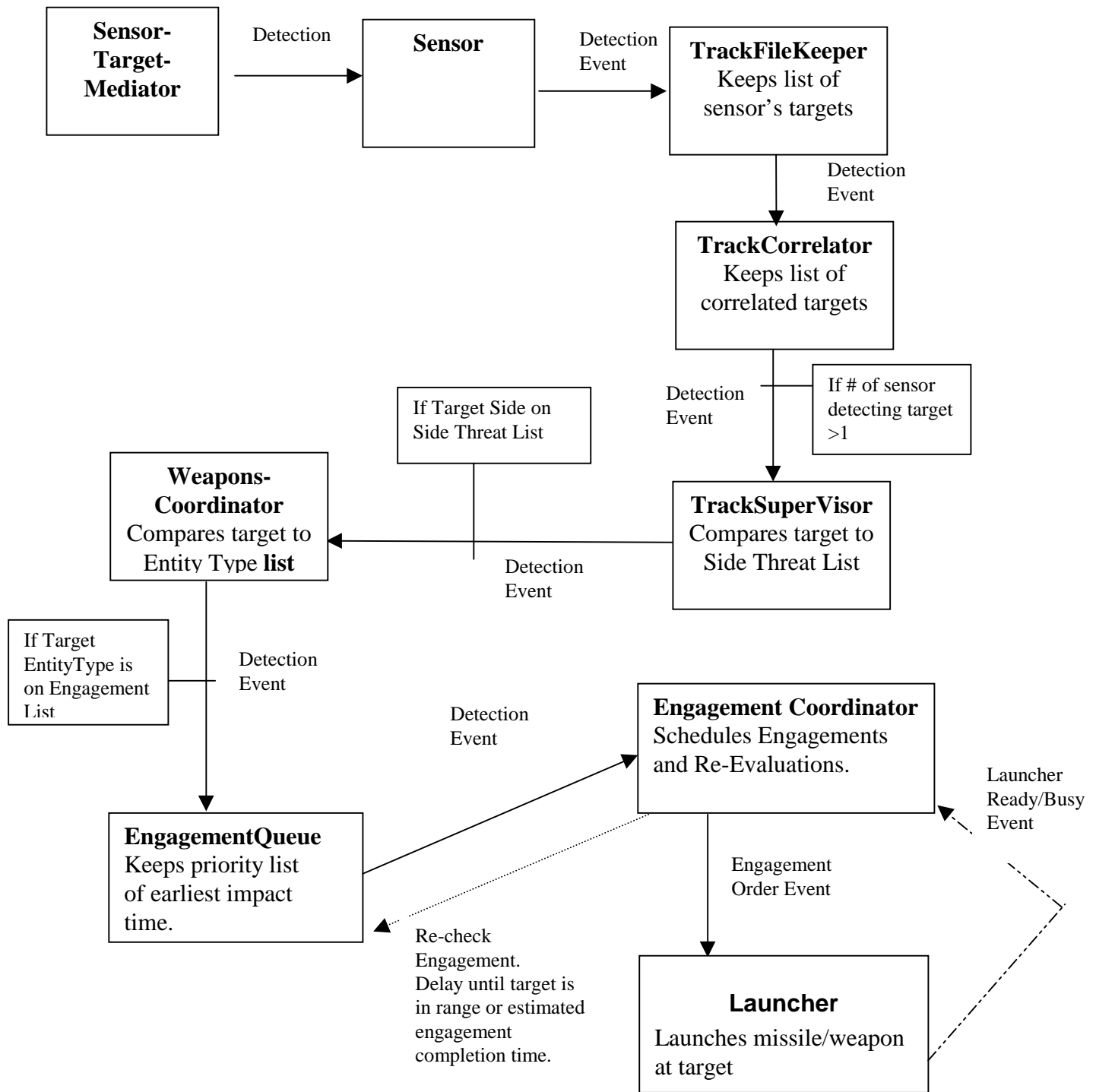
Figure 5. DTE Sequence as executed by the FiringChain

The following text appears within the figure:

Sensor-Target-Mediator

Detection

Sensor

Detection Event

TrackFileKeeper
Keeps list of sensor's targets

Detection Event

TrackCorrelator
Keeps list of correlated targets

If Target Side on Side Threat List

Detection Event

If # of sensor detecting target >1

Weapons-Coordinator
Compares target to Entity Type **list**

Detection Event

TrackSuperVisor
Compares target to Side Threat List

If Target EntityType is on Engagement List

Detection Event

Detection Event

Engagement Coordinator
Schedules Engagements and Re-Evaluations.

Launcher Ready/Busy Event

EngagementQueue
Keeps priority list of earliest impact time.

Re-check Engagement. Delay until target is in range or estimated engagement completion time.

Engagement Order Event

Launcher
Launches missile/weapon at target

52

F. MODELING COMBAT

In the previous section the basic responses of simulation objects was examined and the idea of using components to model a logic train for expected actions was explored. By manipulating the objects different responses could be tested to find the best behavior. Once a course of action has been chosen by a set of components their actions will have interactions with other objects in the simulation. Previously the idea of using sensor Mediators to administrate the interaction between a sensor and a target was illustrated. A combat simulation must also model the way objects destroy, damage or degrade each other.

1. Fly Out

Before a simulation can estimate the effectiveness of a weapon against a target, it must come into contact with the target in some way. This step in a weapons interaction is known as the fly out. If we fire a missile at an aircraft we need to know if it can physically intercept the target. If it is not fast enough, it does not have enough endurance, or the geometry is all wrong, we can see the missile is not able to attack the target. While the EngagementCoordinator may have checked to see if a target was engageable, the circumstances may have changed in flight. Therefore, all missiles have a maximum endurance, a lethal radius and some method of directing its travel. Some head along set way points using the SmoothLinearMover, others pursue a target. There are others that use a combination of the two with a sensor. They fly along a route, upon finding a target, they head towards it for an attack.

Using the SmoothLinearMover and BasicMover as a base, most weapons will be flown out to their targets as part of the simulation. If they can arrive at the target, or close enough, the weapon system will call on the Umpire to adjudicate the outcome. In the case of a missile, it may have to adjust the P(hit) and P(kill) based on the kinematic positioning of the missile and the target at the time of interception.

## 2.	Weapon

The Weapon is essentially a BasicMover that checks to see where it is at the end of its single leg of motion. At the end of this leg, the Weapon will determine if it is within the lethal radius of its intended target, if it is, the Weapon will detonate and attack the target. Either way it will self-destruct at the end of its leg. The Weapon is designed to be used to simulate weapons that head towards the position of the target at release time and are not updated during flight. The Weapon can be used to simulate Anti-Aircraft Artillery, Bombs, and direct fire weapons.

## 3.	Pursuit Missile

The PursuitMissile is based on a PursuitMover which extends the BasicMover. The PusuitMover is designed to continuously maneuver at best speed towards the target's last known position and to periodically update the last known position based on a cycle rate. Therefore, the PursuitMissile will always be in a tail chase position towards its target.

The PursuitMissile is designed to be used to simulate missiles that head directly towards the target. It can be used to simulate Beam Riding Missiles, Track Via Missile Missiles, Semi-Active Radar Homing Missiles, Semi-Active Laser Homing Missiles, Infra-Red Homing Missiles, Electro-Optic Homing Missiles, Imaging Infra-Red Missiles and Passive Radar Homing Missiles. The PursuitMissile can be used to simulate laser guided bombs and smart glider bombs.

## 4.	Interceptor Missile

The InterceptorMissile is based on an InterceptorMover, which extends the BasicMover. The InterceptorMissile is designed to proceed towards an intercept point of the target at its best speed. If the target changes velocities the Interceptor Missile computes a new intercept point. It calculates this intercept position by determining where the target will be by the time the InterceptorMissile could get to the targets current position, then updating the intercept point until the distance between the old point and new point are less than the accuracy of the InterceptorMissile.

The InterceptorMissile is designed to be used to simulate missiles that head towards an intercept point of its intended target. It can be used to simulate Command

Guidance Missiles, Inertial with Mid-Course Guidance and Terminal Active Radar Homing Missiles, and Active Radar missiles.

5.   Cruise Missile

The CruiseMissile is an extension of the SmoothLinearMover. It moves along a pre-set route and attempts to attacks its target after arriving at its last destination. If the target is within its lethal radius then it will call upon the Umpire to evaluate the engagement. If not, it self-destructs. The CruiseMissile is designed to be used to simulate weapons that fly a pre-set route and attack a set postion. The CruiseMissile can be used to simulate a Tomahawk Missile, Joint Stand Off Weapon, Stand-off Land Attack Missile, etc.

6.   Active Cruise Missile

The ActiveCruiseMissile is an extension of the CruiseMissile. It moves along a pre-set route and will turn on a sensor at a preset time. If it detects a target that it likes, it will switch to a Pursuit mode and head towards the target. If it arrives within the lethal radius of the target, it will attack the target.   The ActiveCruiseMissile is designed to be used to simulate weapons that fly a pre-set route and then use a sensor to direct it towards a target. The ActiveCruiseMissile can be used to simulate Anti-Ship Cruise Missiles, Anti-Radiation Missiles, and Area Search Weapons.

G.   MODELING FORCES USING COMPONENT LIBRARY

Using the component library units will be built by composition with additions to represent their particular characteristics and behavior. The general design of the objects is described here. Figure 5 shows the example of a Fighter constructed from components to incorporate all of the properties necessary to model the actions and characteristics in a simulation.
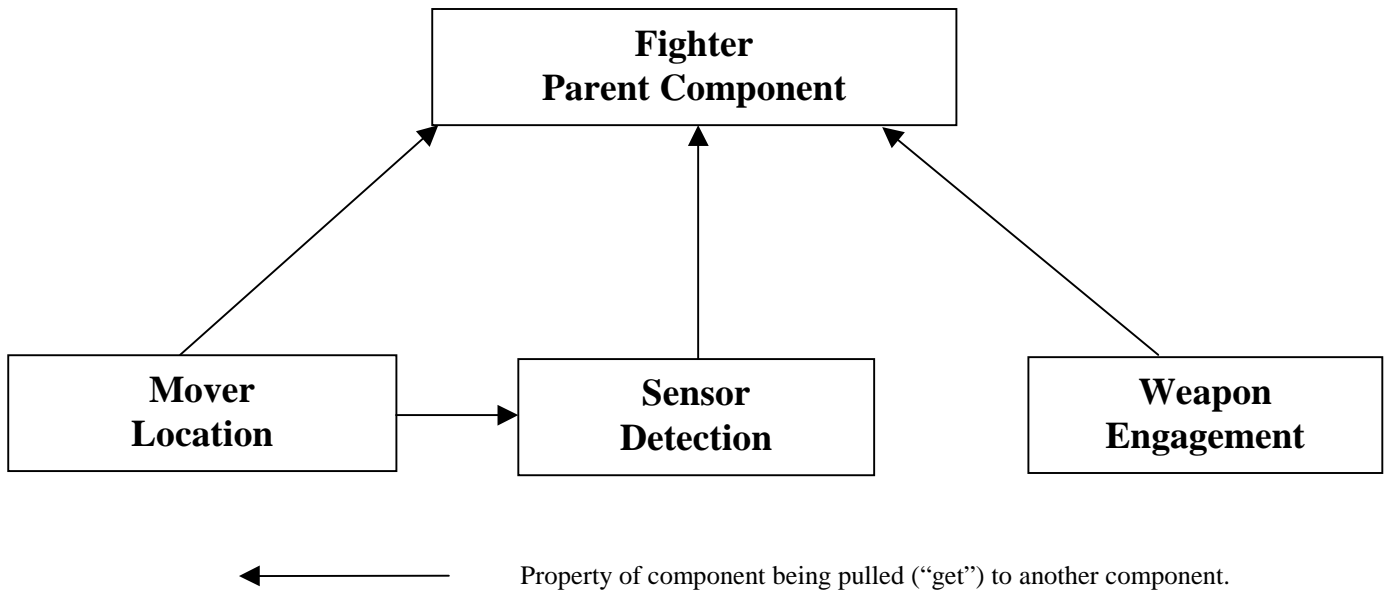
Figure 6. Component Design of Fighter

1.   Bunkers and Control Centers

National Command center bunkers are just extensions of the BasicCombatEntity but are part of the target list that ends the scenario because they would feed into the larger Theater issues the commander is interested in.. But Ground Control Centers and Fire Direction Control Centers are where the DTE sequences for aircraft and theater air defenses occur. If they are destroyed then no direction will be given to those units under their control to intercept or attack enemy forces. Air units and anti-air units can still respond in local mode but will not gain coordination from dead control centers.

2.   Aircraft Shelters

Aircraft shelters themselves are just extensions of the BasicCombatEntity, but when they are destroyed the base will determine if there was an aircraft inside. If so, the base will decrement aircraft from its list. Bases may have both attack aircraft (SU-27) and CAP aircraft (Mirage 2000 and MIG-27) stationed on them.

3.   Air Defense Batteries

Surface to Air Missile batteries use the DTE sequences hooked into their own surveillance sensors and directing launchers. For those batteries with separate vehicles fulfilling certain tasks, the appropriate components will be within appropriate (and separate) vehicles. Generally, there are at least one of each kind of installation or vehicle: Surveillance radar, Fire Distribution Center, and launchers. Theater and Long Range batteries usually have separated vehicles for each part, while Local Air Defense (LAD) batteries combine these functions into one chassis and thus are contained in one unit. Some Air Defense Batteries will be augmented by additional surveillance systems such as passive search and track systems or by coordination from a Fire Direction Control Center.

4.   Air Tower

Air towers are extensions of the BasicCombatEntity and are targets for strike units trying to neutralize air fields. The destruction of an air tower will lengthen the time it takes an air field to deploy fighters.

5.   Anti-Aircraft Artillery

Anti-Aircraft Artillery act in a manner similar to Air Defense Batteries but use dumb weapons to model their bursts of AAA fire instead of missiles. Depending on their type they may be guided by an active radar or optical systems. These include ZSU-23 with 57mm guns.

6.   Fighters

Fighters use the concept of composition to a great degree. First a Mission Manager keeps track of its inbound and outbound wave-points similar to a SmoothLinearMover. When directed to it will use an InterceptMover to move them to intercept an enemy aircraft. When the fighter has completed an intercept the Mission Manager will return it to the nearest point in its current leg and continue on its way.

Fighters will attack enemy units either when directed to by flight leaders or group leaders with its main weapons (Phoenix, AMRAAM, Sparrow) using the leader's DTE sequence. Fighters will automatically fire self defense weapons (Side-Winder) when they detect enemy units within 10 nautical miles using their own DTE sequence. Fighters will

have their own sensors according to their type and the fighter's DTE systems as well as its leader's DTE systems will listen to these. Fighters can be used to model fighter Aircraft such as the F-14D Tomcat, F-15 Eagle and the F-16 fighter as well as other fighters.

7. Attack Aircraft

Attack Fighters will generally have their own sensors and DTE sequences to enable them to attack enemy ground targets, in addition to firing self defense weapons against enemy aircraft. Attack aircraft can be used to model the F/A-18 E/F Super Hornet as well as the EA-6B Prowler and other ground attack aircraft.

8. Bombers

While generic bombers would behave much like an Attack Aircraft, we will use this category for aircraft that fire an OTH weapon towards targets they cannot detect. The Mover Manager in their case notify the weapons system when the Bomber has reached the particular wave-point that calls for an attack to be launched. At this point the Bomber's launchers will start to fire Cruise Missiles based on the mission it has stored for them. In all other respects the bomber behaves like an Attack Aircraft.

9. Airborne Early Warning (AEW) aircraft

Aircraft like the E-2C Hawkeye and the E-3 Sentry have long range active sensors to assist the strike group to detect enemy aircraft. Airborne Early Warning (AEW) aircraft will fly a fixed route and report all targets they detect to the Group Commander, who will decide the response. Generally the Group Commander will be considered inside the AEW aircraft using their Command and Control systems to coordinate with the group. AEW aircraft are their own Flight Commanders and report directly to the Group Commander.

```
                  ┌─────────────────────┐
                  │ Strike Group Chain of│
                  │      Command         │
                  └─────────────────────┘
                            │
                  ┌─────────────────────┐
                  │       Wing          │
                  │     Commander       │
                  └─────────────────────┘
                            │
                  ┌─────────────────────┐
                  │       Group         │
                  │     Commander       │
                  └─────────────────────┘
           ┌────────────────┼────────────────┐
   ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
   │    Flight    │  │    Flight    │  │    Flight    │
   │  Commander   │  │  Commander   │  │  Commander   │
   └──────────────┘  └──────────────┘  └──────────────┘
           ┌────────────────┼────────────────┐
   ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
   │   Aircraft   │  │   Aircraft   │  │   Aircraft   │
   │  Commander   │  │  Commander   │  │  Commander   │
   └──────────────┘  └──────────────┘  └──────────────┘
```
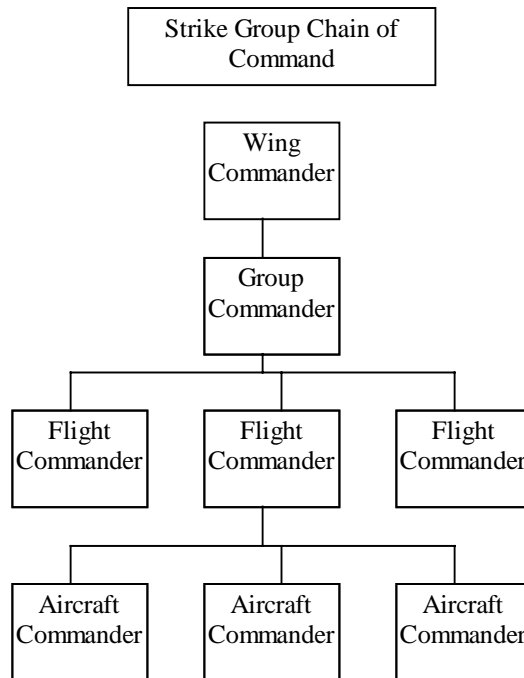
Figure 7. Command and Control

10. Commanders

Commander use the DTE sequence to determine when to attack targets and with
what assets. Commanders also determine when an aircraft is on an inbound or outbound
leg or conducting some other special maneuver such as an interception. Aircraft
Commanders are in charge of their own aircraft. Figure 7 shows the distribution of
responsibility of commanders for forces within their elements. Flight Commanders are in
charge of two to four aircraft of a specific type. Flight Commanders are considered on
any remaining aircraft in their Flight. Group Commanders are in charge of composite of
aircraft Flights conducting a mission. Group Commanders can be on any one aircraft but
generally will be on a Command and Control aircraft.

11. Wing Commanders

Wing Commanders exist at the Aircraft Carrier or air base where the Group
originates. It uses the Adjutant class to keep track of aircraft status to record losses,
recoveries and repairs. The Wing Commander assigns how many of what type of aircraft
will go on a mission.

12.   Airfields

The Airfield is a launcher that launches aircraft on their assigned missions at a constant rate. The Airfield incorporates many of the objects discussed above including aircraft shelters, air tower, Ground Control Interceptor (GCI) station and other assets. The manager records when it loses objects like an aircraft shelter and records the effect, like the reduction in the number of aircraft available.

The manager also controls when aircraft are launched for CAP or for Deck Launched Interception (DLI). The manager will generally have one third of its aircraft on CAP, one third ready for DLI, and the remaining in maintenance.

13.   Installations

Installations such as Airfields, Air Defense Batteries and Command Bunkers may have a standard OOB with their own systems. These will use composition to quickly replicate the whole installation with a single command. In some cases, because the equipment is mobile, elements of the installation, such as LAD batteries, etc. will be randomly placed within a certain distance from the origin of the main installation. For the scenarios incorporated here, they are within six hours of their initial position.

H.   REFEREES

The previous discussion included the branching of the simulation component concept between the "measuring" and measured objects. We use standard objects to represent the behaviors and characteristics of specific physical objects. However we must have a mechanism to adjudicate the interactions of those objects. The Referee object handles those issues for us. Sensor Mover Mediators handle the interaction of sensors and objects and tells the sensor when it detects the object. Umpires handle a specific kind of event between objects. Registrars ensure that the sensors and targets have the proper mediators set up between them. Adjutants keep track of what forces are lost and may trigger the end of a simulation.

```
                    ┌──────────────────────┐
                    │  ModComponent        │
                    │  starting "cell"     │
                    └──────────────────────┘
                              │
             ┌────────────────┴────────────────┐
   ┌──────────────────┐              ┌──────────────────┐
   │  Referees        │              │  Players         │
   │  "Measuring"     │              │  "Measured"      │
   │  Interaction     │              │  Isolated Behavior│
   └──────────────────┘              └──────────────────┘
           │
  ┌────────┼─────────────────┬──────────────────┬──────────────────┐
┌───────────────┐  ┌───────────────┐  ┌───────────────┐  ┌───────────────┐
│ Mediators     │  │ Umpire        │  │ Registrar     │  │ Adjutant      │
│ Component to  │  │ Single        │  │ Matching      │  │ Measuring     │
│ Component     │  │ Interaction   │  │ Mediators     │  │ Conditions    │
└───────────────┘  └───────────────┘  └───────────────┘  └───────────────┘
       │
┌───────────────┐
│ Sub-Mediator  │
│ Component to  │
│ Component     │
└───────────────┘
```
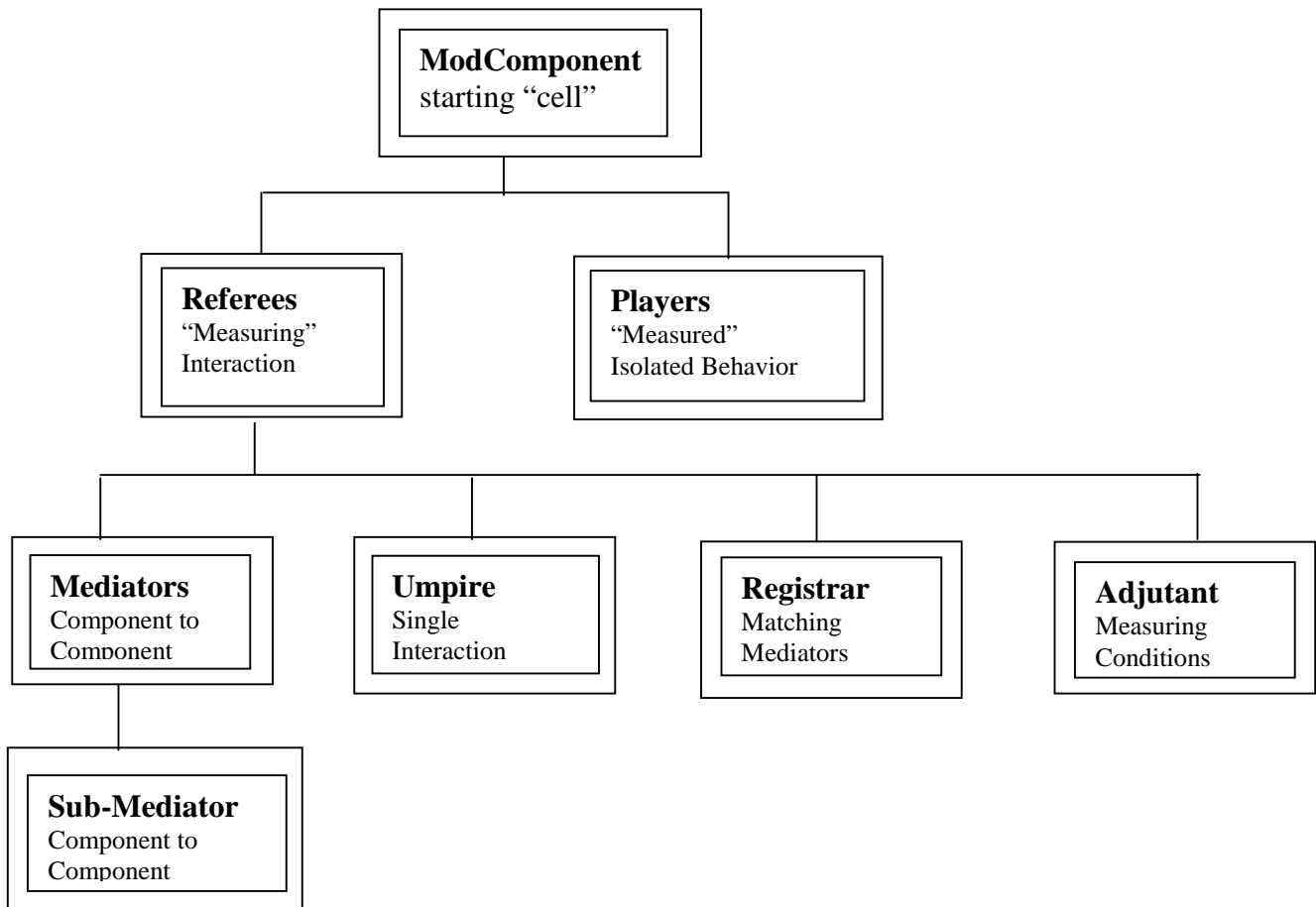
Figure 8. Referees

1. Mediators

Major Arntzen built the first of these Referees as a Mediator.

With these generic components and containers in hand Major Arntzen designed a specialized component that could adjudicate interactions between components. In his example he showed the time it would take a sensor to detect a target. Having constructed a Mediator class, a programmer is then able to re-use the class multiple times and ensure there is a sensor-target-mediator for each sensor and target pair. Instead of one monolithic program to manage all of the potential detections and non-detections within a simulation, the programmer can break them down into bite sized pieces.

2. Sub-Mediators

Just as objects can be broken down into smaller components that handle certain properties and generate or handle certain events, so can the tasks of a Mediator. By taking advantage of the power of composition, we can break the tasks of adjudication down into smaller digestible parts that are easier to solve. In the case of simulating an active radar sensor trying to detect a target, we know that we must take into account the issues of range, elevation and Radar Horizon. We can break these calculations down into their constituent parts and allow Sub-Mediators to handle them. One can solve the issue of range, while another can check the angle to the target or use an altitude band as a substitute, while a third solves for when the two objects will not be blocked by the curvature of the earth. More can be added on later to check terrain and weather effects.

Each Sub-Mediator will report its findings to the main Mediator, who will then make a logic check to see if all agree that the requirements are met to report a detection.

3. Time Master

By modifying the ModComponent in Modkit to extend Professor Buss' SimEntityBase we enabled the BasicModSimComponent to be used in an event driven simulation process. Future simulations can continue using the Schedule class as our time referee or create another Discrete Event Time Master. Either way, this Time Master is a critical referee devoted to one particular task.

4.   Registrar

We have discussed the need to create Mediators between objects to handle their interactions. Each Mediator exists between two objects and handles one aspect of their interaction. An object may have two different kinds of sensors as components. Therefore, there would have to be two kinds of Mediators using different methodology. One sensor may be an active radar that detects the target based on its Radar Cross Section (RCS), while the other is an Infra-Red sensor that is based on the target's Infra-Red Signature (IRS). Two Mediators would be required, one for each sensor-target pair.

Building sufficient Mediators for each run of a simulation can be time consuming and difficult to do on the fly in the midst of a run. Therefore, it is useful to have a Registrar available to hook up the proper Mediator when needed.

As a static and singular object the Registrar is designed, like the Schedule, to be ready for all who call upon it. The Registrar becomes a factory that builds the necessary Mediator to specifications as objects register themselves.

The Registrar keeps a list of all objects in their different categories. As an object is registered, the Registrar adds it to its list and then, based on the object's type, cycles through the appropriate list of objects and creates the necessary mediators between them. In our example we have two types of sensors registered and we register a new target. The Registrar would build an Active Sensor Mediator between the target and the active sensor and then build a Passive Sensor Mediator between the target and the passive sensor. If a future sensor is registered, the Registrar would build the appropriate mediator between the target and the new sensor.

This gives a simulation the ability to build a Mediator on the fly in the middle of a simulation, without having to plan for it in advance.

5.   Umpire

In any simulation or game there are times when a specific, singular event requires adjudication of its outcome. It does not need a mediator to constantly adjudicate two specific objects interaction, only one important event.

The Umpire is designed to adjudicate singular important events between two objects and provide an outcome. This may require the Umpire to set a parameter in one of the objects.

In the game of baseball, the umpire makes the determination of whether the hit is fair or foul and calls the throw a strike or a ball, etc. These are judgment calls that involve issues of chance on singular events.

In a wargame or combat simulation objects will attempt to destroy or degrade each other. When one component attacks another, it has a probability of successfully hitting and then degrading its target. When called upon the Umpire's job is to gain the necessary data from the components, take into account their environment and consults a Data Table. Just as in a wargame the Umpire then throws a die to see if the attack is successful.

The Umpire is a static, singular object that all components can call upon as a universal Referee for combat. When called upon the Umpire can act in a manner similar to the Registrar and direct the issue to a specific method, or the objects involved know which method to call upon directly. The Umpire can then employ its own set of Sub-Umpires/Mediators and/or refer to a CRT before drawing a random number and deciding on the outcome. The outcome may be to deliver a MessageEvent to the victim that he is out as first base, or dead on arrival.

6. Data Table

This class fulfills the role of a CRT. It reads in a text file supplied by the user and saves it in a format that the Umpire can call upon during an engagement. The Data Table provides the Umpire, or other Referees, values agreed upon by the analysts as the probability of an action succeeding. For example, the probability of a missile hitting an aircraft, etc. These values could come from a game, historical accounts, or be the results of high-resolution simulations, etc.

The advantage of this design is the analyst may change the probability of success of an interaction, such as the probability of hitting the target (P_h) or killing (P_k) the target without changing anything within either of the two objects

If no values are available for a specific interaction, then at the first time the interaction occurs, the Data Table will ask the operator to supply the values and record them for the current iterations. If requested it can save those values in the same format it read them in.

7. Adjutant

Simulations are generally designed to determine an outcome of some sort and must not just start, but end when a certain event occurs. In a military simulation, the events occur when an objective is completed, such as the destruction of a target, etc. The simulation must know the ground truth of the status of objects involved to determine if certain conditions have been met.

The Adjutant acts as a scorekeeper. It can be used to end the scenario when the proper conditions are met and be ready to supply necessary data after the run is completed. The Adjutant is designed to be able to listen to all objects under its jurisdiction and record those MessageEvents alerting it to their demise or degradation or some other event. The Adjutant will record this data and be ready to report when called upon. The Adjutant can also be called upon to end the simulation when the conditions have been met.

The Adjutant is set up in a supervisory role above other SideAdjutants to wait for them to report that certain conditions have been met. For example, we may have one SideAdjutant keeping track of targets of an air raid. Another SideAdjutant is keeping track of the air raid forces. The Adjutant is a supervisor over them. The supervisory Adjutant listens for the first to report that all of its targets have been destroyed and the second reports the raid has returned to base. The supervisory Adjutant could end the simulation when both those conditions are met, or when the second Adjutant reports too many planes in the air raid forces have been destroyed to continue, etc.

Upon ending the scenario, queries can be made to the Adjutant about losses, etc. for data collection.

V.   METHODOLOGY

In order to effectively compare the weapons platforms in this thesis, the simulation methodology used must be explored and documented. Issues such as the assumptions made, the manner in which competing forces are simulated, and the data used for determining the outcome of interactions must be available for reproduction of the experiment and possible changes in the future.

With those factors in hand, we must return to the central question:

What does a Theater Commander gain by employing a SOV armed with a CAV in a theater level combat scenario versus conventional weapons systems such as the CVBG and the AEF.

A.   MEASUREMENTS OF EFFECTIVENESS

To answer our question we must find MOEs that can be measured by the computer in the simulation related to the issues of the commander and then estimate them.

There are two MOEs that will be used in the runs:

1.   Time until target(s) destroyed and mission accomplished.
2.   Forces Lost and Personnel Lost.

In the combat scenario selected the target is a bunker. The time until the bunker has been destroyed will be the primary MOE.

The Adjutant will record how many aircraft and what type are lost. When an aircraft is lost, the number of air crew onboard will also be added for the total crew members lost. This is the secondary MOE that will be recorded.

B.   DATA USED

The values set in the CRT and the weapons involved will determine the level of classification of the outputs of the simulation. These numbers also determine the overall validity of the results of the simulation runs.

1. Verification

Verification refers to "The process of determining that a model implementation accurately represents the developer's conceptual description and specifications." (Ritchie, SIMVAL II, 1992).

Extensive work was conducted to verify the code in the simulation. Each individual component was analyzed by a working group and evaluated in a test program. At each level of composition the code was further test for its own veracity and its interaction with other components. Extensive testing was done to ensure that the tasks delegated to components were working properly and that the overall structure of the compositions accomplished their objectives. Any future work with existing components should follow the same pattern to maintain veracity. New components should be fully tested with their own test programs while further compositions should be comprehensively tested as well.

2. Validation

Validation refers to "The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model." (Ritchie, SIMVAL II, 1992).

While the author has attempted to copy the actions of the military systems being modeled by imitation of their doctrine, it cannot be considered fully validated. However, if errors were found, they could be easily be changed given the modular design. If the behavior of the object in question needed to be changed, it could easily be done as well.

The combat portions of the simulation do use sources that are considered validated or have been tested for internal consistency. Specifically, these are the values in the CRT and employed by the Umpire when adjudicating the engagement of a system by a weapon.

3. Open Sources

Unclassified products have been used for the initial runs to test the process. Jane's Defense documents have been used for weapons characteristics and structure. For those values not available in Jane's; such as probability of hit and kill, probability of

detection, etc; values have been taken from Clash of Arms' ***Harpoon***[4] (Bond & Carlson, 1996).

4.   Classified Sources

The Joint Munitions Effectiveness Manual , Air to Surface Weapon Characteristic Manual (JMEM, 1997) CD-ROM has been used at its UNCLAS setting to gain data for the probability of kill for Air to Surface weapons to destroy targets. In some cases the weapons systems in question do not exist, therefore a stand-in has been used.   The penetrator version of the CAV has been modeled using the GBU-24 penetrator data.

Several documents can be used to support classified runs. The JMEM (Air to Surface) values at the SECRET level can be used for Air to Surface values. The JMEM (Anti-Air) can be used at the SECRET setting for ranges, effectiveness and probability of detection for Anti-Air batteries. Anti-Air Batteries design and operation can be taken from the Threat Reference Guide and Counter-tactics manual (a SECRET document). Data from the Survivability Archive For Analysis and Retrieval of Information (SAFARI) manual as well as the JMEM (Air to Air) document can be used for Air to Air weapons engagements data. Both those documents are SECRET.

C.   ASSUMPTIONS

There is only a certain level of detail that can be captured in any simulation. Every simulation model necessarily creates abstractions of the many characteristics of the situation being modeled. However, we can incorporate those necessary bearing weight on the object in the simulation. Other issues below the level of detail or not essential to the question can be left out; but, these must be identified for outside sources to evaluate the results.

The following issues were identified as not being directly related to the question of the thesis and are described below:

1.   Intelligence/Battle Damage Assessment

All strike warfare missions require an assessment of the effectiveness of attacks to determine if the mission was accomplished. There are a number of methods that can be used and combined to determine the effectiveness of an attack: overhead surveillance,

direct observation by friendly forces including Unmanned Arial Vehicle (UAV) vehicles, monitoring signals of the target, Bomb Damage Assessment (BDA) based on tapes of the bomb attack and others. For this thesis, the scenario will assume perfect intelligence regarding the effectiveness of an attack.

The two conventional platforms supporting the mission such as the S-3 Viking, E-3 Sentry and others used to collect data are assumed. For the CAV it is assumed the SOV is available to launch the missions.

The location of fixed forces are known within standard Global Positioning System (GPS) errors and all other forces are known within 6 hours of the beginning of the scenario.

2.   Identification

IFF is assumed to be perfect for both allied and enemy systems. No fratricide will occur and there are no neutral forces on the battle field. Elements within the scenario will get a perfect response to a "Side" query even to weapons in flight.

3.   Morale

Friendly and hostile forces will act according to doctrine and will not "bug-out" because of fear or trepidation. Morale and Esprit de Corps is not modeled, only doctrine. It will not be possible to rout the enemy from the field of battle, only destroy him.

4.   Terrain and environment

Combat will take place on a flat plain. There are no terrain issues to block LOS or hide units in the Area of Operations (AO). The mediators do take into account the effect of the curvature of the earth; however, environmental effects such as rain, cloud cover, smoke and even darkness are not accounted for in this simulation.

VI.   SCENARIO

This chapter will describe the scenario used to test the SOV armed with the CAV against the CVBG and the AEF.

A.   BACKGROUND

Potential opponents of the United States will not stand still and allow our military forces to retain dominance without resistance. Potential opponents could be those we are currently concerned about and others that we have not even considered.

It is necessary to build a credible scenario in which they would be employed to conduct an effective Analysis of Alternatives (AoA) of the SOV, CVBG and AEF. This requires that an adversary be designed with an effective OOB based in realistic facilities. A scenario was built using a 2010 time frame to include weapons systems that are slated to be in the hands of U.S. front line forces.

This scenario will assume an enemy country, called Orange, that borders an ally of the United States, called Green, which does not have U.S. forces stationed in country at the beginning. Orange launches an assault aimed at taking control of Green's resources and territory. The United States responds by sending rapid deployment forces to bolster Green's forces. Additionally, the U.S. will employ a strike package such as a CVBG, AEF or the CAV employed by a squadron of SOVs. Their goal will be to strike critical targets in Orange country enabling forces on the ground to halt the assault.

To enhance credibility, the scenario will model former Soviet Bloc weapons that are currently available for export on the open weapons market. A notional history would be that the United Nations has lifted embargoes against Iraq. Iraq chooses to rebuild its weapons capability by buying weapons from Russia. By 2010 they could have rebuilt their former war machine and pose a serious threat to Saudi Arabia or Kuwait. On the other hand, this same OOB could be used to simulate India, Iran, Pakistan or North Korea in 2010. By using a 2010 time frame, the scenario will enable the use of available deployment models for proposed U.S. weapons systems while realistically restricting the type of potential enemy weapons systems available. This enhances the credibility of the scenario while attempts to project past 2010 are extremely difficult.

The United States would employ its strike weapons against critical targets early in a crisis in an effort to stop the advance of the enemy and then rout it as in the Gulf War. The most important targets would include Command and Control (C^2) facilities, particularly national command authority bunkers, regimental command facilities, ground control interceptor centers (GCICs), etc. No existing models capture the loss of effectiveness for enemy forces, whereas the loss of these assets can severely hamper an enemy's ability to make war. Therefore, the scenario is to attack a heavily defended C^2 facility in a bunker complex.

B. FORCE STRUCTURE

Now that the objectives are identified and the competing systems are known, we must define the enemy opposition and define our competing forces.

1. Enemy forces

In the scenario one the target is a large underground bunker complex. It can be considered a national command authority control center where the head leadership conducts most of their Command, Control, Communications and Intelligence (C^3I) activities.

It is defended by a SA-10 battery, upgraded with several Infra-Red Search and Track (IRST) systems. These will enable it to use both passive and active acquisition methods against potential attackers. Additionally there is an SA-15 Battery.

2. Space Operations Vehicle armed with Common Aero Vehicle

A standard SOV squadron consists of 12 low orbit capable SOVs fitted to launch 3 CAVs per sortie to anywhere in the world. They will be assumed to each conduct one sortie per day with 3 CAVs per sortie.

3. Carrier Battle Group

The inventory of aircraft available to the CVW is described in Chapter I. The following is the "Fair Share" of forces used in a strike package, which enables the CVW to rotate three strike packages continuously.

Four F-14D Tomcat fighters armed with 4 Phoenix air to air missiles, 2 AMRAAM air to air missiles and 2 Sidewinder air to air missiles.

Four F/A-18E Super Hornet fighters loaded for Air to Air missions. They are armed with 4 AMRAAM air to air missiles and 2 Sidewinder air to air missiles.

Four F/A-18E Super Hornet fighters loaded for Surface to Air missions. They are armed with 2 AGM-130 Air to Surface Missiles and 2 Sidewinder air to air missiles.

Four F/A-18F Super Hornet fighters loaded for SEAD with 2 High Speed Anti-Radiation Missiles (HARM) and 2 Sidewinder air to air missiles. They are also equipped with the HARM Tactical System (HTS) which enables them to respond to active radars.

4. Air Force Expeditionary Air Wing

The inventory of platforms available to the AEW is listed in Chapter I.   The following is the "Fair Share" of those forces available to a an AEW strike package to maintain two rotating strike packages:

Four F-15D Eagle fighters loaded for air Superiority missions armed with 4 AMRAAM air to air missiles and 4 Sidewinder air to air missiles.

Four F-15E Strike Eagle fighters loaded for attack missions armed with 4 AMRAAMs and 2 AGM-130 strike weapons.

Four F-16A Falcon Fighters loaded for air superiority weapons armed with 4 AMRAAMs and 2 Sidewinder air to air missiles.

Four F-16 C/J Falcon Fighters loaded for SEAD missions armed with 2 HARM and 2 Sidewinder missiles and equipped with the HTS, enabling it to fire HARMs at active radars.

VII.   ANALYSIS OF RESULTS

The Scenario was simulated as follows. An SOV squadron launched one SOV every two hours. Each SOV was armed with three CAV unitary penetrators. Defending the enemy bunker was one SA-10 battery upgraded with 14 IRST. The normal search Clam Shell radar and the IRST systems were set to queue the Flap Lid fire control radar to engage the CAV.

A series of replications using one random stream for the detection rates and another for the combat resolution Umpire took two hours to run. Forty-five replications total were conducted using the sequential seed of the last random variable of the last run to set the seed of the following run. One 5 run set was conducted followed by four 10 run sets.

The estimated mean time until the Bunker was destroyed was 1.77 hours with an estimated standard deviation of 2.828 hours. The target was almost always destroyed by the end of the second sortie. The most noticeable result was the ineffectiveness of the SA-10 battery. When the Flap Lid sensor gained a track on the CAV and successfully launched a missile, it had a .45 probability of interception. However, the CAV came in so fast that the firing chain rarely had an opportunity to fire its counter batteries. The highest contribution to the over all success of the engagement was the probability of the CAV to hit (.70) minimum damage (.1525) and probability of catastrophic damage for a single hit (.25). The effect of the SA-10 battery was not significant.

The estimated 95 percent confidence interval for time until destruction had a lower limit of .944 hours and an upper limit of 2.957 hours. This means that a Theater Commander can be 90 percent sure that a heavily defended and fortified target can be destroyed within 3 hours of giving the launch command.

VIII.   THEATER LEVEL QUALITATIVE ANALYSIS

The estimated effectiveness of the CAV unitary penetrator against a large, fortified, heavily defended target such as a foreign national C^2 complex needs to be considered in the larger context of a Theater Commander's overall campaign plan. To fully evaluate the impact these results could be used in a larger model to assist a commander in deciding the effectiveness of the SOV/CAV system. Issues to consider include the length of the conflict, the deterrent value to prevent conflicts from starting, and the value of personnel lost or captured due to enemy fire.

A.   TIME AS A FACTOR IN COMBAT

The overall length of a campaign is a significant issue to Operational and Theater Commanders. The monetary and human cost to sustain forces in the region and maintain a high level of readiness for combat operations is significant. The faster a conflict is resolved, the sooner deployed forces can be returned home. In addition to the cost of the military forces involved, the impact to the civilian population and their economy is significant. The longer the conflict continues, the longer their lives are on hold.

Therefore, a Theater Commander will want to use forces that bring a conflict to a close as quickly and effectively as possible. The SOV/CAV's speed of action can greatly assist in the bringing of a conflict to a swift conclusion.

1. C^4ISR
The use of a SOV/CAV against enemy C^2 systems, specifically foreign national C2 facilities will enable Operational Commanders to separate enemy commanders from the mechanisms to control their forces. By bypassing the surrounding enemy defense batteries and striking enemy command bunkers, U.S. forces can disrupt enemy commander's cycle of operations and eliminate his best command centers. This ability would contribute greatly to the U.S. strategy of Battle Space Dominance (Joint Vision 2110, 1998).

2. Blunting of Attack

The use of a SOV/CAV against C^2 systems, logistics centers and other targets would enable an Operational Commander to disrupt an adversary's ability to sustain combat operations. By striking the enemy's rear, without risking U.S. lives, the Operational Commander can blunt an enemy's attack and halt his advance. The Operational Commander could hit these targets within 3 hours of the opening of the assault, blunting it before it is able to seize a significant amount of territory. A potential enemy may be stopped before his forces leave his own territory with sufficient warning.

3. Cost

The longer forces are in the field, the higher the cost to keep them there. The impact of hostile forces in the region will have an additional negative economic consequence to the non-combatants in the region. While cost is never the commander's primary concern, it is an issue that bears examination for decision making.

B. DETERRENT VALUE

With the ability to strike an enemy's infrastructure within three hours of the opening of a conflict, the Operational Commander will have a significant deterrent force available. A potential foe would have to consider his own personal safety inside his national command facilities. He may reconsider the costs and benefits of conducting a military operation knowing that a weapon could hit his bunker complex within 30 minutes of his opponent's decision.

C. AIRCREW

It is impossible to quantify the cost of dead or captured U.S. aircrews on national and military morale. The sight of American helicopter aircrew bodies dragged through the streets of Mogodishu had an incredible impact on the psyche of the American public. While the SOV/CAV cannot replace the use of tactical aircraft in combat, it can greatly assist in the reduction of risk to our pilots by striking fixed targets that are heavily defended and/or fortified. The SOV/CAV may even be effective in destroying semi-mobile forces such as the SA-10 battery by hitting them within 30 minutes of their targeting.

While the simulation conducted here used the unitary penetrator version of the CAV, the other version carries self deploying munitions such as cluster weapons, autonomous attack systems and even Unmanned Aerial Vehicles (UAV). If a UAV is used, a target could be located, attacked and re-evaluated within an hour of the mission authorization. These potential missions merit further assessments and evaluation.

IX.   FINDINGS AND RECOMMENDATIONS

This chapter is devoted to the recommendations of the author in light of the data from the simulations. These recommendations include the future of the SOV/CAV, SBA and the Combat Modeling Simulation Package.

A.   SPACE OPERATIONS VEHICLE/COMMON AERO VEHICLE

Given the significant savings of time and personnel demonstrated in the simulation by the CAV/SOV, it merits further study in the area of strike missions. Further simulations should examine the potential effectiveness of other ordnance and the combination of the SOV with conventional air forces during operations.

The Combat Modeling Simulation Package and its modular design would be an effective tool to use in this analysis. With a flexibly designed tool such as this, the analysts can change the simulation to examine the area of interest, without having to rebuild the entire simulation system. The analysts can switch from evaluating strike missions to others such as satellite replacement, reconnaissance and other missions without having to write a new simulation package each time.

B.   SIMULATION BASED ACQUISITION

If SBA is going to deliver on its promise of evaluating new systems in an iterative fashion at every phase in the acquisition process, it must use flexible, modular and easy to use simulation systems. Modular design simulations such as the Combat Modeling Simulation Package will be essential to the SBA process.

The modular and flexible design is critical for making systems or doctrinal changes in weapons concepts. As data becomes available and more detail is required, it will need to be incorporated into the model. Large fixed simulations cannot do this. SBA cannot succeed without this new or a similar software architecture. New military technologies cannot be properly evaluated and employed without SBA.

C.   FURTHER EVALUATIONS OF STRIKE PACKAGES

The Combat Modeling Simulation package can be used for existing combat systems as well as planned systems. Using JMEMs data, etc. the model can be used to evaluate the effectiveness of strike packages, strike plans, mission sorties, etc. Analysts at the squadron level can construct their particular strike packages, model their doctrine, assemble their targets and evaluate target defenses. Insight can be gained using these simulations as to the effectiveness of strike platforms, TTPs and combinations of weapons systems. The inherent flexibility of Component Based Simulation will grant the analyst the ability to make multiple changes in a short period of time and gain insight into the strengths and weakness of particular courses of action.

X. SUMMARY AND CONCLUSIONS

The original goal of this work was to evaluate the SOV/CAV weapon system against a heavily defended and fortified target and compare it to existing weapons systems. Simulation Based Acquisition requires that all future weapons systems be evaluated in an iterative process where data from field-testing and other inquiries can be incorporated into the simulation and insight gained into their impact.

The challenge was that existing models are primarily designed to model existing systems and do not readily avail themselves to rapid changes or modifications. Therefore a new modeling system had to be developed. The author turned to the new concept of Component Based Simulation to accomplish the modeling.

Next, the purpose and scope of the work was focused to comparing the development and deployment of the SOV/CAV system versus current strike systems with a reasonable level of detail and abstraction.

After narrowing the scope and purpose of this work the current state of Component Based Simulation developed in Modkit was explored. Modkit had to be expanded to incorporate basic combat entities with characteristics of Side, Entity Type, Force Level, Alive and their parent. With this done the existing library was expanded to model motion, sensing & detection, basic logic and behavior. This library was further expanded to model the interaction of objects in combat. With this library more complicated objects such as bunkers, aircraft, the SOV etc. were modeled through composition.

With this set of modeling tools available we explored the methodology used in the simulations. Data used came from unclassified sources, but could easily be replaced by more detailed information. Certain assumptions were made regarding BDA, IFF, morale and the simulation terrain environment.

A combat scenario was developed where an SOV squadron launched one SOV armed with three CAVs per hour against an enemy bunker complex defended by and SA-10 battery. A cluster of 14 IRST systems for queuing its fire control radar augmented the SA-10 battery.

The SOV/CAV combination destroyed the bunker in an average of 1.77 hours. Further qualitative analysis examined the impact of being able to destroy an enemy's command facilities early in a campaign and the possibility of such a weapon system to deter a potential foe.

A Theater CINC would benefit greatly by being able to deploy the SOV/CAV combination against heavily defended and fortified targets. The SOV/CAV system merits further examination in the SBA process using Component Based Simulation models.

Future weapons systems must be evaluated under the SBA process and SBA will require Component Based Simulation models to meet those demands.

APPENDIX A. PRELIMINARY AIR CAMPAIGN ANALYSIS

A.   AIR DEFENSES

To increase the success of a mission and minimize the chances of the loss of aircraft and personnel, a number of missions must be directed towards the destruction or SEAD. Whether a portion of missions are directed to destroy the enemy forces in preliminary missions or suppression assets are devoted during the execution of a mission, the more resources devoted to that end will reduce the number of aircraft expected to be damaged or destroyed. There are a large number of conditional values and permutations that affect this formulation and make them impossible to solve in a mathematical model.

We can use a simple circulation model to explore the difficulties of how many missions to devote to attacking enemy targets or defenses.

Let $A =$ number of enemy air defenses.

Let $A_i =$ number of enemy air defenses at stage i.

Let A_i ' = the number of enemy air defenses at stage i as allied forces depart through stage i.

Let LAD = expected number of hits by Local Air Defenses at the target area.

$LAD = A_{lad} * Ph_{lad}$.

Let Pha = the probability of an enemy air defense to hit an allied aircraft.

Let $H_i =$ the expected number of hits on an allied air group as it transits an area.

$H_i = A_i * Ph_a$.

Let $E[H_t] =$ the expected number of hits on an allied group as it transits through an area makes an attack and exits through the same stage.

$E[H_t] = 2 * sum (A_i * Ph_a) + LAD$

With no suppression by enemy air forces.

Let Pd = probability an allied mission destroys an enemy air defense.

Let Ps = probability an allied mission suppresses an enemy air defense, because the enemy air defense chooses not to engage allied aircraft for fear of destruction.

Let Px_a = probability an allied aircraft mission destroys a target in the target area other than an air defense.

Let S = number of allied missions devoted to suppress enemy air defenses and is capable of attacking them when they fire.

Let D_i = number of allied missions devoted to destroying enemy air defenses at stage i.

Let X = number of allied missions devoted to attacking an enemy target other than air defenses.

Let V = the total number of planes in a mission group.

V = S + X.

For the purposes of discussion we will assume that one aircraft can conduct one of two types of missions, either attack and suppress enemy air defenses or strike the target. Enemy air defenses destroyed at a stage are no longer available to attack the allied forces on their way out or in future missions. On the other hand, aircraft only have enough resources to conduct one mission and so those that make an attack on an air defense are not available to attack them again until they reload. Those that drop their ordnance could return immediately from a stage, assuming that all stages between them and home are empty of air defenses, else they'll need escorts, which drains assets away from the main force.

Let $E[A_1]$ = the expected number of air defenses left after a deliberate attack on stage 1 defenses.

$E[A_1] = A_1 - (D_1 * Pd)$

$E[H_1] = (E[A_1] - (S * Ps)) * Pha$

$E[A_1`] = (E[A_1] - ((1- Ps) E[A_1] * Pd))$

$E[H_1`] =  E[A_1`] - (\text{remaining missions of suppression aircraft} * Ps).$

As we can see, the formula gets very complicated and becomes even more so if we have multiple stages in addition to the Local Area Defenses (LAD) around the target. The planner has three choices which are to destroy the enemy air defenses before attacking your targets (assuming time is available), effectively suppress them during the battle or somehow bypass the stages of air defenses.

We can add the enemy's use of Combat Air Patrols (CAP) to further muddy the waters. In this case, we now have to add air to air missions to the formulation and consider when they would engage the air raid.

If over the horizon (OTH) weapons are used instead of aircraft, the formula changes.

Let Phw = probability of an enemy air defense to hit an allied OTH weapon.

Let Pxw = probability an allied aircraft mission destroys a target in the target area other than an air defense.

$Pha > Phw$ but $Pxa > Pxw$. While the chances of an air defense battery detecting and engaging an OTH weapon is smaller than detecting and engaging an aircraft, weapons dropped from altitude and directed by a pilot have a greater chance of hitting the target than a cruise missile going to where a target is thought to be.

This is another demonstration of the two-stage effect. In this case, we are using a third stage of the weapon to do our bidding farther down range, sparing our pilots the danger of combat. However, bringing in only those elements necessary for the battle at hand has certain limitations.

The best of both worlds would be to incorporate stealth systems that allow the pilot to be on station directing accurate weapons while reducing the Ph to a small number, Phs. The difficulty is that in the future, newer sensors will start to drive these numbers up. Today we have the restriction of operating all our stealth aircraft at night, thus reducing the number of missions that can be done.

The SOV keeps its logistics base very safe, while being able to reach almost any point around the globe within thirty minutes. The SOV is the ultimate in the two-stage concept by maintaining its base facilities within CONUS. Additionally, it bypasses most of the stages of air defenses described earlier by entering enemy air space directly above the target. High altitude air defense systems may be able to defend against it; but, the CAVs small size and high speed give it many of the advantages of stealth weapons. By bypassing the outer ring of defenses, the SOV/CAV combination can devote more sorties directly against the intended target.

B.   TIME TO STATION VS. CYCLE TIME

In addition to the issue of how many missions to devote to the suppression of enemy defenses and Combat Air Patrols (CAP) we have to consider where the first stage of our operations exists. Carrier aircraft fly from the CVBG while Air Force aircraft must fly from a base facility.

1.   CVBG

The CVBGs circulate continuously from their home ports to areas of concern around the world. Because they cannot stay on station indefinitely, it takes time for them to arrive on station. If there is intelligence that something is brewing, they can be moved to a trouble spot as a sign of American resolve and be ready when the action begins. This means not just the National Command Authority who asks "where are the carriers?"

2.   AEW

The Air Expeditionary Wing can be deployed rapidly; but, the AEW requires time to set up at a new base in theater. The AEW can also conduct a strike mission directly from the CONUS, but this creates a much longer cycle time. Either way, they are completely dependent on air base facilities.   In some cases, the base facility may be in the theater of operations (Such as Saudi Arabia in Desert Storm), this creates the issue of protecting those air fields and the host allowing certain missions to occur. If the host nation is the one being attacked, you will most likely gain operating rights. On the other hand, those facilities may be over run. Other air fields may be in an adjacent area (such as air fields on Diego Garcia in Desert Fox); however, they add to the distance the strike force must fly for each mission and multiple missions require transport aircraft or ships to bring in more munitions.

Let C = the cycle time of a force which includes the amount of time to launch a mission, conduct the mission, return to base and be ready for another mission. Caf is the Cycle time for the AEF. Cn is the cycle time of the Navy aircraft.

Let P = the time to prepare a force of aircraft, which can be the time for the carrier arriving on station or the AEF to arrive at a remote air field and setting up. Paf = the preparation time of the AEF. Pn is the preparation time of the Navy aircraft.

If adjacent areas are available for the AEF then Caf (Air Force) is equal to that of Cn (Navy) and Paf << Pn.

If adjacent areas are not available for the AEF then Caf > Cn, but Paf << Pn. In cases where there is no adjacent airfield, the CVBG aircraft would be able to surpass the AEF aircraft in number of missions flown in an ongoing mission environment.

3.   SOV/CAV

The SOV operates directly from its home base in the CONUS. It requires no setup time to either arrive on station or steam into position. It requires no host country permission, only NCA tasking. The cycle time of the SOV/CAV is projected to be one mission every 8 hours.

APPENDIX B. TREATY ISSUES OF WEAPONS IN SPACE

Treaty Implications of the Space Operations Vehicle

The (SOV) does not violate the terms of current treaties ratified by the United States. The Outer Space Treaty, the Strategic Arms Reduction Treaty (START) and other documents prohibit the following activities in space:

- Placement of nuclear weapons or weapons of mass destruction (WMD) in orbit or stationed in outer space.
- Placement of weapons, military bases, installations, or fortifications on celestial bodies.
- Placement of military personnel on celestial bodies for military missions.

The SOV does not fall into any of these categories for the following reasons:

- The SOV/CAV does not use nuclear weapons or any other WMD.
- The SOV does not attain permanent orbit during its missions and the CAV does not remain in orbit.
- The SOV has a limited endurance and is generally not manned during its normal mission operations.
- The SOV cannot be considered a Heavy Bomber under START definitions.
- The SOV as a space lift system does not violate ABM treaty unless ABM systems are deployed aboard it.

The SOV/CAV design does not violate current treaty obligations of the United States and offers a new ability to defend U.S. space assets and deter aggression (Cooper, Hamner & Schriever 1998).

APPENDIX C. PROGRAMMING CODE FOR BASICMOVER

```
/**
* Phillip E. Pournelle Modified the original designed by :
*     @KULAC,Oray
*     @TURAN,Bulent
*        Edited 13 Sept 98 by Phillip E. Pournelle
*        Changes include change of constructor to have
*        initial position as Coor3D instead of Coor4D
*
*         Edited 27 October 1998 by Phillip E. Pournelle
*        Added a setVelocity method to set velocity as a Coor3D
*     towards a general direction.   Changed setMoving so that if
*     the BasicMover is told to get moving but no valid destination
*     it will stop where it is.
*        If you setDestination and it cannot reach the given location
*     by the given time, it will continue towards the last valid
*     location and give a MovingViolationEvent.
*        Edited by Phillip E. Pournelle 28 October 1998
*     Added a setDestination method with signature Coor3D
*     that will have the mover proceed to the location at maxSpeed.
*        Edited 29 October to add Coor3D and Coor2D to setDestination.
*        Edited Wednesday, 02 December, 1998 to add getDestination
*     and make SetVelocity method to set Destination to null.
*        Edited Friday, 01 January, 1999 to add:
*             double setMaxSpeed
*             Coor4D getFutureLocation for Horizon LOS computation
*             double getFutureTime for Horizon LOS computation
*        Edited Thursday, 21 January, 1999 to add:
*             Coor3D setLocation
*        Edited Thursday, 28 January, 1999 to remove
*          generateArrivalAtLocationEvent from constructor.
*          changed doStartMove to stop BM when current position
*     and destination are the same or have no distance between them.
*        Edited Tuesday, 09 February, 1999 to add setMoving(false) to
*     setAlive.
*
**/

package commodsim.movers;
import commodsim.*;
import commodsim.aircraft.*;
import commodsim.doctrine.*;
import commodsim.events.*;
import commodsim.installations.*;
import commodsim.missiles.*;
import commodsim.movers.*;
import commodsim.referees.*;
import commodsim.sensors.*;
import commodsim.vehicles.*;

import simkit.*;
import java.util.*;
import modkit.*;
import modsim.*;
import modsim.combat.*;
import modutil.spatial.*;
import java.beans.*;
```

```java
public class BasicMover extends commodsim.combat.BasicCombatEntity {

    private Coor4D startPosition;
        //location and time at last scheduled position
    private Coor4D destination;//  Destination capable of reaching
    private Coor4D cantReach;// Bogus Destination resulting in
                              // movingViolationEvent
    private Coor3D velocity;//velocity vector at last position
    private double maxSpeed;//max theoretical speed for this mover
    private double currentSpeed; // current speed of the mover
    private double duration;
    private boolean isMoving;

/**
  * Constructor
**/
    public BasicMover(String        name,
                Coor3D        start,
                double        themaxSpeed) {
        super(name);
        startPosition = new Coor4D(start, Schedule.simTime());
        destination   = new Coor4D(start, Schedule.simTime()+
                        (Double.POSITIVE_INFINITY));
        maxSpeed = themaxSpeed;
        velocity = new Coor3D(0.0, 0.0, 0.0);
        isMoving = false;
    }

    public Coor4D getCurrentLocation() {
        double currentTime = Schedule.simTime();
        double deltaTime = currentTime-startPosition.getT();
        if(velocity == null){
            velocity = (new Coor3D(0.0, 0.0, 0.0));
        }
        Coor3D deltaMove = (Coor3D)velocity.scalarMul(deltaTime);
        Coor3D lPos = new Coor3D(startPosition);
            //make 3D version of lastpos
        Coor3D newPos = (Coor3D) lPos.add(deltaMove);
            //find new 3D pos
        Coor4D currentLocation = new Coor4D
            (newPos, startPosition.getT() + deltaTime);
        return new Coor4D (currentLocation);
            //return 4D version
    }

/**
 *     The doStartMove method checks to see where the BasicMover
 *     is currently at, then finds the coordinate difference
 *     between its current position and its destination.
 *     It then finds the unit velocity vector of this difference,
 *     and multiplies it by the currentSpeed to set the new
 *     velocity.  It then calculates the amount of time until it
 *     would arrive based on currentSpeed and schedules the
 *     arrival accordingly.
**/

    public void doStartMove() {
        this.interrupt("ArrivePoint");
        cantReach = null;
        Coor3D np3D;
        np3D = new Coor3D(destination);
```

```
        Coor3D lp3D;
        lp3D = new Coor3D(startPosition);
        velocity = (Coor3D) (lp3D.vectorTo(np3D)).dir();
        if (velocity == null){
            setMoving(false);
        }
        else {
            velocity = (Coor3D) (velocity.scalarMul(currentSpeed));
            isMoving = true;
            duration=startPosition.timeTo
                (new Coor3D(destination), currentSpeed);
            generateVelocityChangedEvent();
            destination = new Coor4D(np3D, Schedule.simTime() + duration);
            waitDelay("ArrivePoint", duration);
        }
    }

    public void doArrivePoint() {
        startPosition = getCurrentLocation();
        this.setMoving(false);
        generateArrivalAtLocationEvent();
    }

    public void setLocation(Coor3D theLocation){
        startPosition = new Coor4D(theLocation, Schedule.simTime());
    }

    public void setMoving (boolean qMoving){
        if ((!qMoving) || (destination == null)) {
            startPosition = getCurrentLocation();
            velocity = new Coor3D(0.0, 0.0, 0.0);
            isMoving = false;
            generateMoverStoppedEvent();
        }

        if (qMoving) {
            setDestination(destination);
            isMoving = true;
        }
        generateVelocityChangedEvent();
    }

    public void setVelocity(Coor3D vel){
        velocity = new Coor3D(vel);
        destination = null;
        isMoving = true;
        generateVelocityChangedEvent();
    }
/**
 *    The setDestination method checks the Coor type and routes them to
 *   the appropriate Coordinate type to set the object on its way.
 *    A  Coor4D will tell it to arrive at a three dimensional space
 *   by the time of the fourth element T.
 *    A  Coor3D will tell it to make to the given point in space at
 *   its best speed.
 *    A  Coor2D will tell it to make to the given point with the Z axis
 *   as zero, at the best possible speed.
 *    Any larger Coor will generate an IllegalArgumentException.
**/
    public void setDestination (Coor destiney){
```

```
        if (Beans.isInstanceOf(destiney, modutil.spatial.Coor4D.class)){
                Coor4D destination = new Coor4D((Coor4D) destiney);
                destination4D(destination);
        }
        else if (Beans.isInstanceOf(destiney,
modutil.spatial.Coor3D.class)){
                Coor3D destination = new Coor3D((Coor3D) destiney);
                destination3D(destination);
        }
/**
*       Need to move Coor2D is in the modutil.spatial package !
**/
        else if (Beans.isInstanceOf(destiney,
modutil.spatial.Coor2D.class)){
                Coor3D destination =
                        new Coor3D(((Coor2D) destiney).getX(),
                                  ((Coor2D) destiney).getY(),
                                     0.0);
                destination3D(destination);
        }
        else {
                throw new IllegalArgumentException
                        ("not capable of handling Coor of this size: "
                         + destiney);
        }
    }


/**
*       The destination4D method checks to see
*        if the 4D location time coordinate can be made by the
*       BasicMover.  If it can then it sets it as its new
*       destination and sets the currentSpeed to arrrive at
*       the requested time.
*       If not, it gives a printed warning and continues on
*       current course.
**/
    protected void destination4D (Coor4D destiney){
        startPosition = getCurrentLocation();
        double distanceToGo = startPosition.distTo(destiney);
        double arrive = destiney.getT();
        double flightTime = arrive - Schedule.simTime();
        double requiredSpeed = distanceToGo / flightTime;
        if(( requiredSpeed <= maxSpeed) && (requiredSpeed >= 0)){
            interrupt("ArrivePoint");
            destination = destiney;
            currentSpeed = requiredSpeed;
            waitDelay("StartMove", 0.0);
        }
        else{
            cantReach = destiney;
            generateMovingViolationEvent();
        }
    }

    protected void destination3D (Coor3D destiney){

        interrupt("ArrivePoint");
        destination = new Coor4D(destiney, Schedule.simTime());
        currentSpeed = (maxSpeed);
        waitDelay("StartMove", 0.0);
```

```java
    }

    public double getMaxSpeed() {
        return maxSpeed;
    }

    public void setMaxSpeed(double speed) {
        maxSpeed = speed;
    }
    public void setAlive(boolean living) {
        super.setAlive(living);
        if(!living){
            setMoving(false);
        }
    }

    public double getCurrentSpeed() {
        return currentSpeed;
    }

    public Coor3D getCurrentVelocity() {
        return new Coor3D(velocity);
    }

    public Coor4D getDestination() {
        return new Coor4D(destination);
    }
    public Coor4D getFutureLocation (){
        return getDestination();
    }
    public double getFutureTime(){
        return getArrivalTime();
    }
    public double getArrivalTime(){
        if (destination != null){
            return destination.getT();
        }
        else{
            return Double.POSITIVE_INFINITY;
        }
    }
    public boolean getMoving() {
        Coor3D origin = new Coor3D(0.0, 0.0, 0.0);
        if ((origin.distTo(velocity)) > 0.0){
                return true;
        }
        else{
                return false;
        }
    }


    public void generateVelocityChangedEvent() {
        ModEvent e=new VelocityChangedEvent
                (this, velocity);
        notifyListeners(e);
    }
    public void generateArrivalAtLocationEvent() {
        ModEvent e=new ArrivalAtLocationEvent
                (this, startPosition, velocity);
        notifyListeners(e);
```

```
    }
    public void generateMoverStoppedEvent() {
        ModEvent e=new MoverStoppedEvent(this);
        notifyListeners(e);
    }

    public void generateMovingViolationEvent(){
        ModEvent e = new MovingViolationEvent(this, cantReach);
        notifyListeners(e);
    }
    public void reset(){
        if (!(resetDone)){
            startPosition = null;
            destination = null;
            cantReach = null;
            velocity = null;
            super.reset();
        }
    }

/**
  * This main tests the Mover
  *
**/
    public static void main(String[] args) {
        JTabbedFrameModEventListener tbf=new
JTabbedFrameModEventListener();
        BasicMover ship = new BasicMover("Ship",new Coor3D(0,0,0),2);
        ship.addModEventListener(tbf);
        Interrogator i1=new Interrogator
                ("Ship-Positions", ship, "CurrentLocation", 0.5);
        i1.addModEventListener(tbf);

        BasicMover frog = new BasicMover("frog",new Coor3D(0,0,0),2);
        frog.addModEventListener(tbf);
        Interrogator i2=new Interrogator
                ("frog-Positions", frog, "CurrentLocation", 0.5);
        i2.addModEventListener(tbf);

        ship.setProperty("Destination", ( new Coor4D(4,4,4,4)));
        frog.setProperty("Destination", ( new Coor3D(0,0,0)));
        Schedule.setVerbose(true);
        Schedule.setSingleStep(true);
        Schedule.startSimulation();
    }
}
```

APPENDIX D. PROGRAMMING CODE FOR COOKIE CUTTER 3DSENSOR MOVER MEDIATOR

```
/**
* @author Phillip E. Pournelle
*
* @version 0.1
*
* Started Friday, 04 December, 1998
*
*     This NewCookieCutter3DSensorMoverMediator is designed to
*   take a NewCookieCutter3DSensor and a mover and report when the
*   mover is detected and undetected  by the sensor.
*     This mediator uses two sub-Mediators to find out when the
*   object is within the geometry of the sensor.
*   The AltitudeSubMediator will tell when the mover is below the
*   CeilingZ and above the FloorZ.  The RangeSubMeditor will tell
*   when the mover is within and outside the range of the sensor.
*   This mediator will then setDetection of the target on the
*   sensor only when both conditions are met and setUndetection
*   when one of the two conditions are removed.
*     Edited Saturday, 23 January, 1999 to add LineOfSight ability.
*
**/

package commodsim.referees;
import commodsim.*;
import commodsim.aircraft.*;
import commodsim.doctrine.*;
import commodsim.events.*;
import commodsim.installations.*;
import commodsim.missiles.*;
import commodsim.movers.*;
import commodsim.referees.*;
import commodsim.sensors.*;
import commodsim.vehicles.*;

import simkit.*;
import java.util.*;
import java.beans.*;
import modkit.*;
import modsim.*;
import modutil.spatial.*;

public class NewCookieCutter3DSensorMoverMediator extends
                      NewBasicSensorMoverMediator {

//     protected  NewCookieCutter3DSensor sensor;
       protected  RangeSubMediator     rangeSubMediator;
       protected  AltitudeSubMediator altitudeSubMediator;
       protected  LOSSubMediator        lineOfSightSubMediator;
       protected  boolean                inRange;
       protected  boolean                inAltitude;
       protected  boolean                inLOS;

/**
*      Constructor
```

```
**/
    public NewCookieCutter3DSensorMoverMediator
        (String name, NewCookieCutter3DSensor theSensor,
         BasicModSimComponent theTarget){

        super(name, theSensor, theTarget);
        inRange = false;
        inAltitude = false;
        inLOS = false;
        createSubMediators();
    }
/**
*       This method continues the constructor and builds the SubMediators
*   required to determine when the target is within the sensor's
geometry.
*       This method is overwritten in extensions to replace the
SubMediators
* with specific types based on the needs (ActiveSensor, PassiveSensor,
etc.)
**/
    public void createSubMediators(){
            rangeSubMediator = new RangeSubMediator
                (this.getProperty("Name") + "rangeSubMediator",
                sensor, target, this);
            altitudeSubMediator = new AltitudeSubMediator
                (this.getProperty("Name") + "AltitudeSubMediator",
                sensor, target, this);
            lineOfSightSubMediator = new LOSSubMediator
                (this.getProperty("Name") + "LOSSubMediator",
                sensor, target, this);
    }

/**
*       This method is overwritten to do nothing as the subMediators
*   are doing the real lifting to determine when the target is
*   within the geometry of the sensor.
**/
    public void checkGeometry(){
        double blank = 0.0;
    }


/**
*       When either of the SubMediators determines that the target is
*   within their geometry for the sensor it triggers its own InRange
*   method.  If both are true then we go on to the doInRange method.
*   In some extenstions of this class, being in range is just the first
*   step and other methods take over.  Here, we immediately go to the
*   detection step.
**/

    public void setInRange(boolean withinRange){
        inRange = withinRange;
        if ((inRange) && (inAltitude) && (inLOS)){
            waitDelay("InGeometry", 0.0);
        }
        if(!(inRange)){
            if(inGeometry){
                waitDelay("OutGeometry", 00.0);
            }
        }
```

```java
    }

    public void setInAltitude(boolean withininAltitude){
        inAltitude = withininAltitude;
        if ((inRange) && (inAltitude) && (inLOS)){
            waitDelay("InGeometry", 0.0);
        }
        if(!(inAltitude)){
            if(inGeometry){
                waitDelay("OutGeometry", 00.0);
            }
        }
    }
    public void setInLOS(boolean withinLOS){
        inLOS = withinLOS;
        if ((inRange) && (inAltitude) && (inLOS)){
            waitDelay("InGeometry", 0.0);
        }
        if(!(inLOS)){
            if(inGeometry){
                waitDelay("OutGeometry", 00.0);
            }
        }
    }


/**
 *      In this method we immediately go from within geometry to
detection.
 *  This is why it is a CookieCutterSensor.
**/
    public void doInGeometry(){
        if(!(inGeometry)){
            inGeometry = true;
            System.out.println(target + " within Geometry of " + sensor);
            waitDelay("Detection", 0.0);
        }
    }
    public void doDetection() {
        if (tracking == false){
            System.out.println(sensor + " detects " + target);
            tracking = true;
            sensor.setDetection(target);
        }
    }
/**
 *      In this method we not only remove this Mediator as a
ModEventListener
 *  but tell the SubMediators to do the same.  This is how we disconnect
 *  the mediators and pray the garbage collecter notices that they aren't
 *  talking to anyone anymore...
**/
    public void disConnect() {
        super.disConnect();
        rangeSubMediator.disConnect();
        altitudeSubMediator.disConnect();
        lineOfSightSubMediator.disConnect();
        rangeSubMediator = null;
        altitudeSubMediator = null;
        lineOfSightSubMediator = null;
```

```
    }
    public static void main (String [ ] args) {
        JTabbedFrameModEventListener tbf =  new
JTabbedFrameModEventListener();
        SmoothLinearMover Smog = new SmoothLinearMover
            ("Smog", new Coor3D(0.0, 0.0, 0.0),
            new Coor4D[] {
                new Coor4D(1.0, 1.0, 1.0, 1.0),
                new Coor4D(2.0, 2.0, 2.0, 2.0),
                new Coor4D(3.0, 3.0, 3.0, 3.0),
                new Coor4D(4.0, 4.0, 4.0, 4.0),},
                2.0);
        NewCookieCutter3DSensor SmogLight = new NewCookieCutter3DSensor
         ("SmogLight", Smog, true, 1.0, -1.0, 1.0);
        BasicMover Frodo = new BasicMover
            ("Frodo", new Coor3D(2.0, 2.0, 2.0), 0.01);
        NewCookieCutter3DSensorMoverMediator SmogLightMediator1 =
            new NewCookieCutter3DSensorMoverMediator
            ("SmogLightMediator1", SmogLight, Frodo);
        Interrogator i1 = new Interrogator
            ("Smog-Position", Smog, "CurrentLocation", 1.0);
//      Interrogator i2 = new Interrogator
//          ("Frodo-Position", Frodo, "CurrentLocation", 1.0);
        Smog.addModEventListener(tbf);
        SmogLight.addModEventListener(tbf);
        i1.addModEventListener(tbf);
//      i2.addModEventListener(tbf);

        Schedule.setVerbose(true);
        Schedule.setSingleStep(true);
        Schedule.startSimulation();
    }

}
```

```java
/**
* @author Phillip E. Pournelle
*
* @version 0.1
*
* Started Friday, 04 December, 1998
*
*      This RangeSubMeditor works with the
*   NewCookieCutter3DSensorMoverMediator and is designed to
*   take a NewCookieCutter3DSensor and a mover and report when the
*   mover is within the range geometry of the sensor based
*   on its maximum range
*      The RangeSubMeditor will tell
*   when the mover is within and outside the range of the sensor.
*   This mediator will then set the InRange Boolean on the Mediator
*   for the sensor.
*
**/

package commodsim.referees;
import commodsim.*;
import commodsim.aircraft.*;
import commodsim.doctrine.*;
import commodsim.events.*;
import commodsim.installations.*;
import commodsim.missiles.*;
import commodsim.movers.*;
import commodsim.referees.*;
import commodsim.sensors.*;
import commodsim.vehicles.*;

import simkit.*;
import java.util.*;
import modkit.*;
import modsim.*;
import modutil.spatial.*;

public class RangeSubMediator extends NewBasicSensorMoverMediator{
      protected static int count;

      protected NewBasicSensorMoverMediator boss;

   public RangeSubMediator
      (String name, NewBasicSensor theSensor,
            BasicModSimComponent theTarget,
            NewBasicSensorMoverMediator theBoss){
      super(name, theSensor, theTarget);
      count = count + 1;
      boss = theBoss;
   }

/**
*      This method is overwritten to determine the times of when
*   the target is within range and leaves the range of the sensor
*   based on the maximum range of the sensor.  It uses the relative
*   motion of the sensor and target.
**/

   public void checkGeometry(){

      double[ ] times;
```

```java
        this.interrupt("InRange");
        this.interrupt("OutOfRange");
        times = getRangeTimes(target, sensor);
        if (times  != null) {
            if (times.length == 2){
                double targetArrivalTime =
                    ((Number)
target.getProperty("ArrivalTime")).doubleValue();
                double sensorArrivalTime =
                    ((Number)
sensor.getProperty("ArrivalTime")).doubleValue();
                if (((times[0] + Schedule.simTime()) <= targetArrivalTime)
                    && ((times[0] + Schedule.simTime()) <=
sensorArrivalTime)){
                    if (!(tracking)){
                        waitDelay("InRange", times[0]);
                    }
                }
                if (((times[1] + Schedule.simTime()) <= targetArrivalTime)
                    && ((times[1] + Schedule.simTime()) <=
sensorArrivalTime)){
                    if (tracking == true){
                        waitDelay("OutOfRange", times[1]);
                    }
                }

            }
            else{
                Coor4D targetPosition =
                    (Coor4D)(target.getProperty("CurrentLocation"));
                Coor4D sensorPosition =
                    (Coor4D) (sensor.getProperty("CurrentLocation"));
                double distance = targetPosition.distTo(sensorPosition);
                double maxRange = determineMaxRange();
                //  This is for when the target is within range but with
                // velocity of zero.
                if (distance <= maxRange){
                    if (tracking == false){
                        waitDelay("InRange", 00.0);
                    }
                }
                else {
                    if (tracking){
                        waitDelay("OutOfRange", 00.0);
                    }
                }
            }
        }
    }

/**
 *    This method uses Professor Buss' Quadratic Equation solver
 *  to determine when the target would be crossing the ranges of the
 *  sensor's maximum range.
**/


    public double[ ] getRangeTimes
        (BasicModSimComponent target, NewBasicSensor sensor){

        double maxRange = determineMaxRange();
```

```java
        Coor3D sensorPosition = new Coor3D
          ((Coor4D) sensor.getProperty("CurrentLocation"));
        Coor3D sensorVelocity =
          (Coor3D) sensor.getProperty("CurrentVelocity");
        Coor3D targetPosition = new Coor3D
          ((Coor4D) target.getProperty("CurrentLocation"));
        Coor3D targetVelocity =
          (Coor3D) target.getProperty("CurrentVelocity");
        Coor3D origin = new Coor3D(0.0, 0.0, 0.0);
        Coor3D relativeVelocity =
            (Coor3D)targetVelocity.sub((Coor3D)sensorVelocity);
        double velocityNorm = relativeVelocity.norm();

          Coor3D relativePosition =
              (Coor3D)targetPosition.sub((Coor3D)sensorPosition);
          double positionNorm = relativePosition.norm();
          double innerProduct =
relativeVelocity.dotProd(relativePosition);
          double velocitySquared = (velocityNorm * velocityNorm);
          double positionSquared = (positionNorm * positionNorm);
          double maxRangeSquared = (maxRange * maxRange);
          double times[ ] = simkit.smd.Quadratic.solve (velocitySquared,
              2.0 * innerProduct, positionSquared - maxRangeSquared);
          if ((times.length == 2) && (times[1] > 0.0)){
              times[0] = Math.max(times[0], 0.0);
              return times;
          }
          if (times.length == 1) {
              return times;
          }

        return null;
    }

/**
*     When the CheckGeometry determines that the target is within range
*   of the sensor's geometry, it sets the InRange Boolean on the
Mediator.
**/

    public void doInRange(){
        if (tracking == false){
          System.out.println(target + " within range of " + sensor);
          System.out.println("RangeSubMediator" + count);
          tracking = true;
          boss.setProperty("InRange", new Boolean(tracking));
        }
    }
    public void doOutOfRange () {
        if (tracking == true){
          System.out.println(target + " out of Range of " + sensor);
          tracking = false;
          boss.setProperty("InRange", new Boolean(tracking));
        }
    }

    public double determineMaxRange() {
        double maxRange =
              ((Number) sensor.getProperty("MaxRange")).doubleValue();
        return maxRange;
    }
```

```
    public void disConnect(){
        boss = null;
        super.disConnect();
    }
    public void reset(){
        boss = null;
        super.reset();
    }
}
```

```
/**
* @author Phillip E. Pournelle
*
* @version 0.1
*
* Started Friday, 04 December, 1998
*
*       This AltitudeSubMediator works with the
*    NewCookieCutter3DSensorMoverMediator and is designed to
*    take a NewCookieCutter3DSensor and a mover and report when the
*    mover is within the Altitute band of the the sensor based
*    on its FloorZ and CeilingZ and the target's relative motion to
*    the sensor.
*       The RangeSubMeditor will tell
*    when the mover is within and outside the altitude band of the
*    Sensor. This sub mediator will then set the setinAltitude
*    Boolean on the Mediator for the sensor.
*
**/

package commodsim.referees;
import commodsim.*;
import commodsim.aircraft.*;
import commodsim.doctrine.*;
import commodsim.events.*;
import commodsim.installations.*;
import commodsim.missiles.*;
import commodsim.movers.*;
import commodsim.referees.*;
import commodsim.sensors.*;
import commodsim.vehicles.*;

import simkit.*;
import java.util.*;
import modkit.*;
import modsim.*;
import modutil.spatial.*;

public class AltitudeSubMediator  extends NewBasicSensorMoverMediator{

        protected NewBasicSensorMoverMediator boss;

    public AltitudeSubMediator
        (String name, NewBasicSensor theSensor,
             BasicModSimComponent theTarget,
             NewBasicSensorMoverMediator theBoss){
        super(name, theSensor, theTarget);
        boss = theBoss;
    }

/**
*       This method is overwritten to determine the times of when
*    the target is within the Altitude band of the sensor
*    based on the CeilingZ and FloorZ of the sensor
*       The FloorZ and CeilingZ are based on the relative position
*    of the sensor and the relative motion between sensor and target.
**/

    public void checkGeometry(){
        double[ ] times;
        this.interrupt("InBand");
```

```java
        this.interrupt("OutOfBand");
        times = getRangeTimes(target, sensor);
        if (times  != null) {
            if (times.length == 2){
                double targetArrivalTime =
                    ((Number)
target.getProperty("ArrivalTime")).doubleValue();
                double sensorArrivalTime =
                    ((Number)
sensor.getProperty("ArrivalTime")).doubleValue();
                if (((times[0] + Schedule.simTime()) <= targetArrivalTime)
                    && ((times[0] + Schedule.simTime()) <=
sensorArrivalTime)){
                        if (tracking == false){
                            waitDelay("InBand", times[0]);
                        }
                }
                if (((times[1] + Schedule.simTime()) <= targetArrivalTime)
                    && ((times[1] + Schedule.simTime()) <=
sensorArrivalTime)){
                        if (tracking == true){
                            waitDelay("OutOfBand", times[1]);
                        }
                }
            }
        }
        //  This is for when the target is within altitude band
        // but with vertical velocity of zero.
        else {
            Coor4D targetPosition =
                (Coor4D)(target.getProperty("CurrentLocation"));
            double targetAltitude = targetPosition.getZ();
            Coor4D sensorPosition =
                (Coor4D) (sensor.getProperty("CurrentLocation"));
            double sensorAltitude = sensorPosition.getZ();
            double relativeAltitude = targetAltitude - sensorAltitude;
            double floor = ((Number)
                (sensor.getProperty("FloorZ"))).doubleValue();
            double ceiling = ((Number)
                 (sensor.getProperty("CeilingZ"))).doubleValue();
            if ((relativeAltitude <=  ceiling) && (relativeAltitude >=
floor)){
                        if (tracking == false){
                            waitDelay("InBand", 0.0);
                        }

            }
        }
    }

/**
 *      This method finds when the relative motion between the sensor
 *   and target would cause the target to cross the sensor's altitude
 *   bands.  If both times is less than zero it returns null.  If
 *   only one is below zero it returns zero and the positive number
 *   It always return the lesser time first.
**/

    public double[ ] getRangeTimes
        (BasicModSimComponent target, NewBasicSensor sensor){
        Coor4D target4DPosition = (Coor4D)
```

```java
            target.getProperty("CurrentLocation");
        double targetAltitude = target4DPosition.getZ();
        Coor4D sensor4DPosition = (Coor4D)
            sensor.getProperty("CurrentLocation");
        double sensorAltitude = sensor4DPosition.getZ();
        Coor3D targetVelocity = (Coor3D)
            target.getProperty("CurrentVelocity");
        double targetZvelocity = targetVelocity.getZ();
        Coor3D sensorVelocity = (Coor3D)
            sensor.getProperty("CurrentVelocity");
        double sensorZvelocity = sensorVelocity.getZ();
        double velocityDifference = targetZvelocity - sensorZvelocity;
        double relativeAltitude = targetAltitude - sensorAltitude;
        if (velocityDifference == 0.0){
            return null;
        }
        double maxAlt =
            ((Number)(sensor.getProperty("CeilingZ"))).doubleValue();
        double minAlt =
            ((Number)(sensor.getProperty("FloorZ"))).doubleValue();
        double maxTime =
            (sensorAltitude + maxAlt - targetAltitude)/velocityDifference;
        double minTime =
            (sensorAltitude + minAlt - targetAltitude)/velocityDifference;
        if ((minTime <= 0.0) && (maxTime <= 0.0)){
            return null;
        }
        maxTime = Math.max(0.0, maxTime);
        minTime = Math.max(0.0, minTime);
        double startTime = Math.min(maxTime, minTime);
        double endTime = Math.max(maxTime, minTime);
        double[] Times = new double[] {startTime, endTime};
        return Times;
    }

/**
 *     When the CheckGeometry determines that the target is within range
 *   of the sensor's geometry, it sets the InAltitude Boolean on the
Mediator.
**/

    public void doInBand(){
        System.out.println(target + " within Altitude Band of " + sensor);
        tracking = true;
        boss.setProperty("InAltitude", new Boolean(tracking));
    }
    public void doOutOfBand () {
        System.out.println(target + " out of Altitude Band of " + sensor);
        tracking = false;
        boss.setProperty("InAltitude", new Boolean(tracking));
    }
    public void disConnect(){
        boss = null;
        super.disConnect();
    }
    public void reset(){
        boss = null;
        super.reset();
    }
}
```

```
/**
* @author Phillip E. Pournelle
*
* @version 0.1
*
* Started Friday, 04 December, 1998
*
*      This LOSSubMeditor works with the
*   NewCookieCutter3DSensorMoverMediator and is designed to
*   take a NewCookieCutter3DSensor and a mover and report when the
*   mover is within the Line of Sight geometry of the sensor based
*   on the curvature of the Earth.
*      The LOSSubMeditor will tell
*   when the mover is within and outside the LOS of the sensor.
*   This mediator will then set the LOS Boolean on the Mediator
*   for the sensor.
*
**/

package commodsim.referees;
import commodsim.*;
import commodsim.aircraft.*;
import commodsim.doctrine.*;
import commodsim.events.*;
import commodsim.installations.*;
import commodsim.missiles.*;
import commodsim.movers.*;
import commodsim.referees.*;
import commodsim.sensors.*;
import commodsim.vehicles.*;

import simkit.*;
import java.util.*;
import modkit.*;
import modsim.*;
import modutil.spatial.*;

public class LOSSubMediator extends NewBasicSensorMoverMediator{
      protected double HORIZON_CONSTANT = 1.22;
      protected boolean lineOfSight;

      protected NewBasicSensorMoverMediator boss;

   public LOSSubMediator
      (String name, NewBasicSensor theSensor,
            BasicModSimComponent theTarget,
            NewBasicSensorMoverMediator theBoss){
      super(name, theSensor, theTarget);
      boss = theBoss;
      lineOfSight = false;
   }

   public boolean checkLOS
      (BasicModSimComponent target, NewBasicSensor sensor){

      Coor4D sensorPosition = new Coor4D
        ((Coor4D) sensor.getProperty("CurrentLocation"));
      Coor4D targetPosition =
```

```java
        (Coor4D)(target.getProperty("CurrentLocation"));
        double sensorHeight;
        sensorHeight = ((Number)
            target.getProperty("SensorHeight", new
Double(0.0))).doubleValue();

        double sensorAltitude = sensorPosition.getZ();
        double targetAltitude = targetPosition.getZ();
        double LOSRange = HORIZON_CONSTANT *
            (Math.sqrt(sensorAltitude + sensorHeight) +
             Math.sqrt(targetAltitude));

        Coor2D sensor2DPosition = new Coor2D(sensorPosition);
        Coor2D target2DPosition = new Coor2D(targetPosition);
        double arcDistance = sensor2DPosition.distTo(target2DPosition);

        if (arcDistance < LOSRange){
            return true;
        }
        else {return false;}
    }

    public void doCheckGeometry(){
        checkGeometry();
    }

/**
 *      This method is overwritten to determine the times of when
 *  the target is within range and leaves the range of the sensor
 *  based on the maximum range of the sensor.  It uses the relative
 *  motion of the sensor and target.
**/

    public void checkGeometry(){
        this.interrupt("CheckGeometry");
        lineOfSight = checkLOS(target, sensor);
        if ((tracking) && (!(lineOfSight))){
            waitDelay("OutOfRange", 0.0);
        }
        if ((!(tracking)) && (lineOfSight)){
            waitDelay("InRange", 0.0);
        }
        tracking = lineOfSight;
        double[] times;
        times = getRangeTimes(target, sensor);
        double targetArrivalTime =
                ((Number)
target.getProperty("ArrivalTime")).doubleValue();
        double sensorArrivalTime =
                ((Number)
sensor.getProperty("ArrivalTime")).doubleValue();
        if (((times[0] + Schedule.simTime()) <= targetArrivalTime)
                && ((times[0] + Schedule.simTime()) <=
sensorArrivalTime)){
            if(times [0] != Double.POSITIVE_INFINITY){
                waitDelay("CheckGeometry", times[0] + 0.1);
            }
        }
    }

/**
```

```
*      This method determines when the units will gain Line Of Sight
*  of each other and returns the time it will occur.
*
**/

    public double[ ] getRangeTimes
        (BasicModSimComponent target, NewBasicSensor sensor){

//      double[] timeToCheck;  // the time that we will return to
checkGeometry.
        double timeOne;  // time of LOS based on sensor vertical motion.
        double timeTwo;  // time of LOS based on target vertical motion.
        double timeThree;// time of LOS based on relative motion.
        double timeFour; // Minimum of (timeOne, timeTwo, & timeThree).
        double timeFive; // time of LOS based on combination of motions.
        double timeSix;  // Minumum of (timeFour and timeFive).
        double sensorHeight;
        double sensorAltitude;
        double sensorHorizonRange;
        double sensorZVelocity; // Vertical component of sensor velocity
        double targetHeight;
        double targetAltitude;
        double targetHorizonRange;
        double targetZVelocity; // Vertical component of target velocity

    /**
      *    Set up values for calculations
    **/
        Coor4D sensorPosition = new Coor4D
          ((Coor4D) sensor.getProperty("CurrentLocation"));
        sensorHeight = ((Number)
          target.getProperty("SensorHeight", new
Double(0.0))).doubleValue();
        sensorAltitude = sensorPosition.getZ();
        sensorHorizonRange = HORIZON_CONSTANT *
            (Math.sqrt(sensorAltitude + sensorHeight));
        sensorZVelocity = ((Coor3D)
          sensor.getProperty("CurrentVelocity")).getZ();

        Coor4D targetPosition = new Coor4D
          ((Coor4D) target.getProperty("CurrentLocation"));
        targetHeight = ((Number)
          target.getProperty("targetHeight", new
Double(0.0))).doubleValue();
        targetAltitude = targetPosition.getZ();
        targetHorizonRange = HORIZON_CONSTANT *
            (Math.sqrt(targetAltitude));
        targetZVelocity = ((Coor3D)
          target.getProperty("CurrentVelocity")).getZ();

        Coor2D sensor2DPosition = new Coor2D(sensorPosition);
        Coor2D target2DPosition = new Coor2D(targetPosition);
        double arcDistance = sensor2DPosition.distTo(target2DPosition);

    /**
      *  Find timeOne, time until sensor's vertical motion would
      *  gain LOS.
    **/

        timeOne = (arcDistance - targetHorizonRange)/HORIZON_CONSTANT;
        timeOne = (timeOne * timeOne) - (sensorAltitude + sensorHeight);
```

```
        timeOne = timeOne/sensorZVelocity;
        if (timeOne <=0){
            timeOne = Double.POSITIVE_INFINITY;
        }

/**
  *  Find timeTwo, time until target's vertical motion would
  *  gain LOS.
**/

        timeTwo = (arcDistance - sensorHorizonRange)/HORIZON_CONSTANT;
        timeTwo = (timeTwo * timeTwo) - (targetAltitude + targetHeight);
        timeTwo = timeTwo/sensorZVelocity;
        if (timeTwo <=0){
            timeTwo = Double.POSITIVE_INFINITY;
        }
        timeFour = Math.min(timeOne, timeTwo);

/**
  *      This portion checks for timeThree, the time in which relative
  *  2D motion would bring them into LOS.
**/

        double LOSRange = HORIZON_CONSTANT *
            (Math.sqrt(sensorAltitude + sensorHeight) +
             Math.sqrt(targetAltitude));
        double maxRange = LOSRange;
        Coor2D sensor2DVelocity = new Coor2D
          ((Coor3D) sensor.getProperty("CurrentVelocity"));
        Coor2D target2DVelocity = new Coor2D
          ((Coor3D) target.getProperty("CurrentVelocity"));
        Coor2D origin = new Coor2D(0.0, 0.0);
        Coor2D relativeVelocity =
            (Coor2D)target2DVelocity.sub((Coor2D)sensor2DVelocity);
        double velocityNorm = relativeVelocity.norm();

        Coor2D relativePosition =
            (Coor2D)target2DPosition.sub((Coor2D)sensor2DPosition);
        double positionNorm = relativePosition.norm();
        double innerProduct = relativeVelocity.dotProd(relativePosition);
        double velocitySquared = (velocityNorm * velocityNorm);
        double positionSquared = (positionNorm * positionNorm);
        double maxRangeSquared = (maxRange * maxRange);
        double times[ ] = simkit.smd.Quadratic.solve (velocitySquared,
            2.0 * innerProduct, positionSquared - maxRangeSquared);
        timeThree = Double.POSITIVE_INFINITY ;
        if (times.length == 2) {
            if((times[0] <= 0) && (times[1] <= 0)){
                timeThree = Double.POSITIVE_INFINITY ;
            }
            if(times[0] > 0){
                timeThree = times[0];
            }
            else if(times[1] > 0){
                timeThree = times[1];
            }
        }
        timeFour = Math.min(timeFour, timeThree);
/**
  *        This portion checks for timeFive, the time the combination
  *  motions would bring about LOS.  We don't bother if all the
```

```
  *    contributing factors are POSITIVE_INFINITY.
**/
      if (timeFour == Double.POSITIVE_INFINITY){
          double[] timeToCheck = {timeFour, Double.POSITIVE_INFINITY};
          return timeToCheck;
      }
      else {
          double targetHorizonRate =
              Math.sqrt(targetAltitude + (targetZVelocity * timeFour));
          targetHorizonRate =
              targetHorizonRate - Math.sqrt(targetAltitude);
          targetHorizonRate = targetHorizonRate *
HORIZON_CONSTANT/timeFour;

          double sensorHorizonRate =
              Math.sqrt(sensorAltitude + sensorHeight +
                  (sensorZVelocity * timeFour));
          sensorHorizonRate =
              sensorHorizonRate - Math.sqrt(sensorAltitude +
sensorHeight);
          sensorHorizonRate = sensorHorizonRate *
HORIZON_CONSTANT/timeFour;

          double distanceOne = origin.distTo(relativePosition);
          double distanceTwo = origin.distTo
              ((Coor2D) relativePosition.add((Coor2D) relativeVelocity));
          double closureRate = distanceOne - distanceTwo;
          double LOSrate =
              targetHorizonRate + sensorHorizonRate + closureRate;
          timeFive = arcDistance - (sensorHorizonRange +
targetHorizonRange);
          timeFive = timeFive / LOSrate;
          if (timeFive <=0) {
              timeFive = Double.POSITIVE_INFINITY;
          }
          timeSix = Math.min(timeFour, timeFive);
          double[] timeToCheck = {timeSix, Double.POSITIVE_INFINITY};
          return timeToCheck;
      }
    }

/**
 *    When the CheckGeometry determines that the target is within range
 *   of the sensor's geometry, it sets the InLOS Boolean on the Mediator.
**/

  public void doInRange(){
        System.out.println(target + " within LOS of " + sensor);
        System.out.println("LOS SubMediator");
        boss.setProperty("InLOS", new Boolean(tracking));
  }
  public void doOutOfRange () {
        System.out.println(target + " out of LOS of " + sensor);
        System.out.println("LOS SubMediator");
        boss.setProperty("InLOS", new Boolean(tracking));
  }
  public void handleVelocityChangedEvent(ModEvent e){
      this.interrupt("CheckGeometry");
      checkEnabled();
  }
  public void disConnect () {
```

```java
        this.interruptAll();
        boss = null;
        super.disConnect();
    }
    public void reset(){
        boss = null;
        super.reset();
    }
}
```

LIST OF REFERENCES

Ackerman, G.C., *Responding to the Threat From Third World Air Defense Systems: A Comparison of U.S. Policy Options*, Naval Postgraduate School Monterey, California, December 1990.

Arntzen, A., *Software Components for Air Defense Planning*, Naval Postgraduate School Monterey, California, September 1998.

Bradley, G.H., Buss A.H., *An Architecture for Dynamic Planning Systems Using Loosely Coupled Components,* Naval Postgraduate School, Monterey, California, 1997.

Cooper H., Hamner, R. Schriever, *Military Spaceplane: Weapons in Space, Implications for AFSPC and Possible Advocacy Strategies,* Air Force Research Laboratory, Kirtland Air Force Base, Albuquerque New Mexico, October 1998.

Coyle, P.E., Sanders, P., *Simulation, Test, and Evaluation Process: STEP Guidelines,* Office of the Undersecretary of Defense for Aquisition and Technology, 4 December 1997.

Crawford, K.R., Hatton, M.T., Melton, A.W., *Where Are the Littoral Warfare Fast-Attack Craft?,* Article in U.S. Naval Institue Proceedings, Naval Institue Press, Annapolis, Maryland, April 1995.

Cullen, T., *Jane's Land-Based Air Defense,* Eleventh Edition, 1998-99.

Defense Modeling and Simulation Organization(DMSO)*, DOD Modeling & Simulation Master Plan*, Department of Defense, October 1995

Friedman, G., *The Future of War, Power, Technology and American World Dominance in the 21'st Century,* Crown Publishers, New York, 1996.

Hughes, W. P., *Fleet Tactics: Theory and Tactics*, Naval Institue Press, Annapolis, Maryland 1986.

Hughes, W.P., *Comment and Discussion*, Article in U.S. Naval Institue Proceedings, Naval Institue Press, Annapolis, Maryland, May 1995.

Kulac, O. *A Comparative Analysis of Active and Passive Sensors in Anti-Air Warfare Area Defense Using Discrete Event Simulation Components*, Naval Postgraduate School Monterey, California, March 1999.

Macfadzen, R.H.M., *Surfaced Based Air Defense System Analysis*, Artech House, Boston, Massachusetts 1992.

Military Space Plane Program Executive Officer, *Military Spaceplane Handbook for Wargamers,* Air Force Research Laboratory, Kirtland Air Force Base, Albuquerque, New Mexico, 11 May 1998.

Military Space Plane Program Executive Officer, *System Requirements Document For A Space Operations Vehicle,* Philips Air Force Research Laboratory, Kirtland Air Force Base, Albuquerque, New Mexico, 2 April 1998.

Nicholas, T., Rossi, R., *U.S. Military Aircraft Data Book*, Twentieth Edition, Data Search Associates, 1998.

O'Leary, A.P. *Jane's Electro-Optic Systems,* Third Edition, 1997-98.

Possony, Pournelle, J.E., *The Strategy of Technology,* University Press of Cambridge, Massachusetts, 1970.

Randolph, R., *Deep Target Penetration Vehicle Study,* Naval Air Warfare Center Weapons Division, China Lake, California. October 1993.

Ritchie, A.E., *Simulation Validation Workshop Proceedings (SIMVAL II),* Institute for Defense Analysis, Alexandria, VA, April 1992.

United States Air Force, *Air Force Magazine,* March 1997.

Warhola, P.J., *An Analysis of Alternative Methods To Conduct High-Resolution Activities in a Variable-Resolution Simulation*, Naval Postgraduate School Monterey, California, September 1997.

Williams, S. M., *Synergy In the Joint Conventional Strike Force,* Naval Postgraduate School Monterey, California, March 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center ……………………………………………………………2
     8725 John J. Kingmand Road Suite 9444
     Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library……………………………………………………………………………2
     Naval Postgraduate School
     411 Dyer Road
     Monterey, California 93943-5101

3. Professor Arnold H. Buss………………………………………………………………………2
     Code OR/SB
     Naval Postgraduate School
     Monterey, California 93943-5101

4. Professor Gordon H. Bradley ……………………………………………………………………1
     Code OR/BZ
     Naval Postgraduate School
     Monterey, California 93943-5101

5. Professor Thomas H. Hoivik ……………………………………………………………………1
     Code OR/LA
     Naval Postgraduate School
     Monterey, California 93943-5101

6. Lieutenant Colonel Charles H. Shaw, III………………………………………………………1
     Code OR/SC
     Naval Postgraduate School
     Monterey, California 93943-5101

7. Lieutenant Phillip E. Pournelle ………………………………………………………………3
     12051 Laurel Terrace Drive
     Studio City, California 91604

8. Military Spaceplane Program Office……………………………………………………………3
     3550 Aberdeen Avenue SE
     Kirtland AFB, New Mexico 87117